全国高等职业教育计算机类规划教材·实例与实训教程系列
国家高职示范院校核心课程教材

C 语言实用教程

白羽　刘畅　刘苗苗　主编
袁鸿雁　刘辉　叶宾　胡艳梅　副主编

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书系统地介绍了 C 语言编程知识，共分十二章，内容包括：C 语言概述，C 语言的数据描述与基本操作，C 语言的流程控制，数组，函数，编译预处理，指针，结构体、共用体和枚举类型，位运算，文件，库函数及应用，以及上机实训。

本书注重基础，突出应用，采用案例式教学方法，先举实例，再对相关知识点进行讲解，然后通过“练一练”来总结、熟悉本讲知识点，最后通过“想一想”提出本次课的一些思考题，以便于读者能够更好地理解 C 语言的知识，提高实际编程能力。

本书易教易学、学以致用、注重能力，对初学者容易混淆的内容进行了重点提示和讲解。本书适合作为高职高专类各相关专业的程序设计教材，也适合编程开发人员培训、自学使用。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目(CIP)数据

C 语言实用教程 / 白羽主编.—北京：电子工业出版社，2009.2
全国高等职业教育计算机类规划教材．实例与实训教程系列
ISBN 978-7-121-07969-6
I. C… Ⅱ. 白… Ⅲ. C 语言－程序设计－高等学校：技术学校－教材Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2009）第 005805 号

策划编辑：左　雅
责任编辑：左　雅
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1 092　1/16　印张：20.75　字数：531 千字
印　　次：2009 年 2 月第 1 次印刷
印　　数：4 000 册　定价：29.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

C 语言是近年来国内外广泛使用的计算机程序设计语言，也是软件开发人员必须掌握的一种语言。全国各类高等学校中普遍开设 C 语言课程，全国计算等级考试二级和三级中也包括 C 语言的考试内容。

本书作为 C 语言程序设计的入门与实用教材，共分 12 章。主要内容包括：第一章 C 语言概述，主要介绍程序设计的基本概念、C 语言的特点、C 语言程序的基本结构。第二章数据描述与基本操作，主要介绍 C 语言的基本数据类型、运算符和表达式。第三章 C 语言的流程控制，主要介绍结构化程序设计的三种结构的各种语句格式及功能。第四章数组，主要介绍一维数组、二维数组和字符数组及其应用。第五章函数，主要介绍函数定义及调用、变量和函数的作用域。第六章编译预处理，主要介绍宏定义、文件包含和条件编译三种命令的格式。第七章指针，主要介绍指针的概念、指针的定义、指针的运算，及指针与数组、指针与函数的关系。第八章结构体、共用体和枚举，主要介绍 C 语言的构造数据类型。第九章位运算，主要介绍各种位运算符及其应用。第十章文件，主要介绍文件的基本操作和使用规则。第十一章库函数及应用，主要介绍常用的字符屏幕函数及图形函数。第十二章上机实训，设计一个学生管理系统，可以针对 C 语言编程知识进行综合训练。

本书注重基础，突出应用，采用案例式教学方法，先举实例，再对相关知识点进行讲解，然后通过“练一练”来总结、熟悉本讲知识点，最后通过“想一想”提出本次课的一些思考题，以便于读者能够更好地理解 C 语言的知识，提高实际编程能力。本书易教易学、学以致用、注重能力，对初学者容易混淆的内容进行了重点提示和讲解。

本书配有电子课件，方便教师授课，并提供书中所有思考题、练习题和课后编程题的程序源代码，方便教师授课和读者自学。本书中的所有源程序均在 Turbo C 2.0 环境下通过上机调试。

本书由白羽、刘畅、刘苗苗担任主编，袁鸿雁、刘辉、叶宾、胡艳梅担任副主编。其中第十章由白羽编写，第六、八、十一、十二章及附录 B 和 C 由刘畅编写，第一、二、九章由刘苗苗编写，第四章由袁鸿雁编写，第七章由刘辉编写，第三章由叶宾编写，第五章由胡艳梅编写。全书由刘畅负责统稿。

本书荣获教育部“2009 年高职高专计算机教指委优秀教材”。

由于作者水平有限，经验不足，编写时间仓促，书中难免存在许多缺点及不足之处，恳请广大读者批评指正。

编　　者

序

20 世纪 90 年代以来，以计算机和通信技术为推动力的信息产业在我国获得前所未有的发展，全国各企事业单位对信息技术人才求贤若渴，高等教育计算机及相关专业毕业生供不应求。随后几年，我国各高等院校、众多培训机构相继开设计算机及相关专业，积极扩大招生规模，不久即出现了计算机及相关专业毕业生供大于求的局面。纵观近十年的就业市场变化，计算机专业毕业生经历了“一夜成名、求之不得”的宠幸，也遭遇了“千呼百应、尽失风流”的冷落。

这个时代深深地镌刻着信息的烙印，这个时代是信息技术人才尽情展示才能的舞台。目前我国的劳动力市场，求职人数过剩，但满足企业要求的专业人才又很稀缺。这种结构性的人才市场供求矛盾是我国高等教育亟待解决的问题，更是“以人为本，面向人人”为目标的职业教育不可推卸的责任。

电子工业出版社，作为我国出版职业教育教材最早的出版社之一，是计算机及相关专业高等职业教材重要的出版基地。多年来，我们一直在教材领域为战斗在职业教育第一线的广大职业院校教育工作者贡献着我们的力量，积累了丰富的职业教材出版经验。目前，计算机专业高等教育正处于发展中的关键时期，我们有义务、有能力协同全国各高等职业院校，共同探寻适合社会发展需要的人才培养模式，建设满足高等职业教育需求的教学资源——这是我们出版“全国高等职业教育计算机类规划教材·实例与实训教程系列”的初衷。

关于本系列教材的出版，我们力求做到以下几点：

（1）面向社会人才市场需求，以培养学生技能为目标。工学结合、校企结合是职业教育发展的客观要求，面向就业是职业教育的根本落脚点。本系列教材内容体系的制定是广大高职教育专家、一线高职教师共同智慧的结晶。我们力求教材内容丰富而不臃肿、精简而不残缺，实用为主、够用为度。

（2）面向高职学校教师，以方便教学为宗旨。针对每个课程的教学特点和授课方法，我们为其配备相应的实训指导、习题解答、电子教案、教学素材、阅读资料、程序源代码、电子课件、网站支持等一系列教学资源，广大教师均可从华信教育资源网（www.huaxin.edu.cn）免费获得。

（3）面向高职学校学生，以易学、乐学为标准。以实例讲述理论、以项目驱动教学是本系列教材的显著特色。这符合现阶段我国高职学生的认知规律，能够提高他们的学习兴趣，增强他们的学习效果。

这是一个崭新的开始，但永远没有尽头。高等职业教育教材的建设离不开广大职业教育工作者的支持，尤其离不开众多高等职业院校教师的支持。我们诚挚欢迎致力于职业教育事业发展的有识之士、致力于高等职业教材建设的有才之士加入到我们的队伍中来，多批评，勤点拨，广结友，共繁荣，为我国高等职业教育的发展贡献我们最大的力量！

电子工业出版社高等职业教育分社

目　录

第一章　C 语言概述 …… (1)
第一讲　C 语言基础知识 …… (1)
一、程序设计概述 …… (1)
二、C 语言简介 …… (4)
练一练 …… (11)
想一想 …… (11)
本章小结 …… (11)
课后习题一 …… (11)
第二章　数据描述与基本操作 …… (13)
第二讲　基本数据类型、变量与常量 …… (13)
练一练 …… (21)
本讲小结 …… (22)
想一想 …… (22)
第三讲　运算符与表达式、数据类型的转换 …… (22)
一、算术运算与赋值运算 …… (22)
二、关系运算、逻辑运算与条件运算 …… (26)
三、圆括号运算符、逗号运算符和 sizeof 运算符 …… (29)
练一练 …… (32)
本讲小结 …… (32)
想一想 …… (33)
第四讲　数据的输入与输出 …… (33)
一、字符输入、输出函数 …… (33)
二、标准输入、输出函数 …… (34)
练一练 …… (41)
本讲小结 …… (42)
想一想 …… (42)
本章小结 …… (43)
课后习题二 …… (43)
第三章　C 语言的流程控制 …… (46)
第五讲　选择结构 …… (46)
一、基本 if 语句 …… (46)
二、标准 if 语句 …… (48)
三、复合 if 语句 …… (49)
四、if 语句的嵌套 …… (51)
五、switch 语句 …… (52)
练一练 …… (53)

本讲小结……（55）
第六讲　循环结构……（55）
一、while 语句 ……（56）
二、do-while 语句 ……（57）
三、for 循环语句……（58）
练一练……（60）
本讲小结……（61）
想一想……（61）
第七讲　循环语句的嵌套和流程转向语句 ……（61）
一、循环语句的嵌套……（61）
二、流程转向语句 goto 语句 ……（64）
三、break 语句 ……（65）
四、continue 语句……（66）
练一练……（67）
本讲小结……（69）
本章小结……（69）
课后习题三……（69）
第四章　数组 ……（74）
第八讲　一维数组……（74）
练一练……（77）
本讲小结……（79）
想一想……（79）
第九讲　二维数组……（80）
练一练……（82）
本讲小结……（83）
想一想……（84）
第十讲　字符数组与字符串 ……（84）
练一练……（90）
本讲小结……（91）
想一想……（91）
本章小结……（91）
课后习题四……（92）
第五章　函数 ……（95）
第十一讲　函数定义、调用、函数原型及函数返回语句 ……（95）
一、函数的定义、调用及函数返回语句 ……（95）
二、函数原型 ……（101）
练一练……（102）
本讲小结……（104）
想一想……（104）
第十二讲　函数的嵌套、递归调用及函数之间的数据传递……（104）

一、函数的嵌套调用 ……（105）
二、递归函数及递归调用 ……（107）
三、实参-形参之间的数据传递（值传递方式） ……（109）
四、实参-形参之间的数据传递（数组作函数参数） ……（110）
练一练 ……（112）
本讲小结 ……（115）
想一想 ……（115）
第十三讲　变量作用域及存储类型、内部函数和外部函数 ……（115）
一、作用域和生存期 ……（116）
二、局部变量的作用域和存储类型 ……（117）
三、全局变量的作用域、存储类型及多文件程序的运行 ……（120）
四、内部函数与外部函数 ……（123）
练一练 ……（125）
本讲小结 ……（126）
想一想 ……（126）
本章小结 ……（126）
课后习题五 ……（127）
第六章　编译预处理 ……（132）
第十四讲　宏定义、文件包含和条件编译 ……（132）
一、不带参数的宏定义 ……（132）
二、带参数的宏定义 ……（134）
三、文件包含处理 ……（136）
四、条件编译 ……（138）
练一练 ……（139）
想一想 ……（141）
本章小结 ……（141）
课后习题六 ……（141）
第七章　指针 ……（144）
第十五讲　指针概述与指针赋值、指针的运算 ……（144）
一、指针概述与指针赋值 ……（144）
二、指针的运算 ……（147）
练一练 ……（150）
本讲小结 ……（151）
想一想 ……（152）
第十六讲　指针与数组（一） ……（152）
一、一维数组元素的指针访问方式 ……（152）
二、二维数组元素的指针访问方式 ……（154）
三、字符指针与字符串 ……（156）
练一练 ……（157）
想一想 ……（159）

本讲小结……（159）
第十七讲　指针与数组（二）……（159）
一、指向一维数组的指针……（160）
二、指针数组……（162）
练一练……（164）
想一想……（165）
本讲小结……（165）
第十八讲　指针与函数……（165）
一、指针作为函数参数……（165）
二、指针函数……（168）
三、指向函数的指针……（169）
四、带参数的 main 函数及其应用……（172）
练一练……（173）
本讲小结……（175）
想一想……（175）
本章小结……（175）
课后习题七……（176）
第八章　结构体、共用体和枚举……（181）
第十九讲　结构体基础……（181）
练一练……（188）
本讲小结……（189）
想一想……（189）
第二十讲　结构体数组和指向结构体的指针……（189）
一、结构体数组及指向结构体变量的指针……（189）
二、指向结构体数组的指针……（192）
练一练……（194）
本讲小结……（196）
想一想……（196）
第二十一讲　结构体与函数……（196）
一、结构体类型的变量作为函数参数……（196）
二、结构体类型的变量作为函数的返回值……（198）
练一练……（199）
本讲小结……（200）
想一想……（200）
第二十二讲　链表……（200）
一、链表基础知识及动态分配函数……（200）
二、链表的操作……（202）
练一练……（207）
本讲小结……（209）
想一想……（209）

第二十三讲　共用体、枚举、typedef 类型定义 ……（209）
一、共用体类型的定义与使用 ……（209）
二、枚举类型的定义与使用 ……（213）
三、typedef 类型的定义及应用 ……（215）
练一练 ……（216）
本讲小结 ……（217）
想一想 ……（217）
本章小结 ……（217）
课后习题八 ……（217）
第九章　位运算 ……（222）
第二十四讲　位运算 ……（222）
一、位运算 ……（222）
二、位段 ……（226）
练一练 ……（228）
想一想 ……（230）
本章小结 ……（230）
课后习题九 ……（230）
第十章　文件 ……（233）
第二十五讲　文件概述、文件打开与关闭 ……（233）
练一练 ……（238）
本讲小结 ……（238）
想一想 ……（238）
第二十六讲　文件读写 ……（239）
一、fputc 函数和 fgetc 函数 ……（239）
二、fputs 函数和 fgets 函数 ……（240）
三、fwrite 函数和 fread 函数 ……（242）
四、fprintf 函数和 fscanf 函数 ……（244）
练一练 ……（247）
本讲小结 ……（248）
想一想 ……（248）
第二十七讲　文件的定位和文件的检测 ……（248）
一、文件的定位函数 ……（248）
二、文件的检测函数 ……（251）
练一练 ……（254）
本讲小结 ……（255）
想一想 ……（255）
本章小结 ……（255）
课后习题十 ……（255）
第十一章　库函数及应用 ……（258）
第二十八讲　字符屏幕处理函数 ……（258）

一、字符窗口的定义……（258）
二、字符窗口的输入/输出函数……（261）
三、字符窗口的屏幕操作函数……（263）
练一练……（265）
本讲小结……（266）
想一想……（266）
第二十九讲　图形处理函数（一）……（266）
一、图形模式的初始化……（266）
二、屏幕颜色的设置和清屏函数……（269）
三、基本画图函数……（272）
练一练……（276）
本讲小结……（276）
想一想……（276）
第三十讲　图形处理函数（二）……（277）
一、基本图形的填充及填充方式的设定……（277）
二、任意封闭图形的填充……（280）
练一练……（281）
本讲小结……（282）
想一想……（282）
第三十一讲　图形操作函数……（282）
一、图形窗口操作……（282）
二、图形模式下的字符……（285）
练一练……（289）
本讲小结……（290）
想一想……（290）
本章小结……（290）
课后习题十一……（290）
第十二章　上机实训……（291）
第三十二讲　学生成绩管理系统……（291）
一、问题描述……（291）
二、数据结构……（291）
三、程序流程……（291）
四、完整程序……（292）
五、程序运行步骤及结果……（296）
附录 A　课后习题参考答案……（298）
课后习题一参考答案……（298）
课后习题二参考答案……（298）
课后习题三参考答案……（299）
课后习题四参考答案……（300）
课后习题五参考答案……（301）

课后习题六参考答案……（303）
课后习题七参考答案……（304）
课后习题八参考答案……（306）
课后习题九参考答案……（307）
课后习题十参考答案……（308）
课后习题十一参考答案……（309）
附录 B　常用字符与 ASCII 码对照表……（311）
附录 C　运算符的优先级、结合方向及口诀……（313）
参考文献……（315）

第一章　C语言概述

本章首先介绍了算法和程序的基本概念、算法流程图和 N-S 盒图，以及结构化程序设计的概念；其次介绍了 C 语言的特点、C 语言程序的特点；最后介绍 Turbo C 2.0 集成环境下的上机操作过程，以及在 Turbo C 2.0 下使用汉字的方法。学习本章的目的是使读者对 C 语言和程序设计有一个概略的了解，并掌握上机运行简单程序的操作步骤。

第一讲　C语言基础知识

学习目标

- 简单了解程序设计的基本思路;
- 掌握 C 语言的发展及特点;
- 掌握 C 语言上机操作步骤，熟悉 Turbo C2.0 集成环境。

一、程序设计概述

小讲堂

1．程序设计的基本概念

1）程序

用计算机语言描述的算法称为计算机程序，或简称**程序**。只有用计算机语言描述的算法才能在计算机上执行，换言之，只有计算机程序才能在计算机上执行。人们编写程序之前，为了直观或符合人的思维方式，常常先用其他方式描述算法，然后再翻译成计算机程序。

2）程序设计及程序设计语言

人类社会中有多种语言交流工具，每种语言又都有它的语法规则。人和计算机通信需要通过计算机语言。计算机语言是面向计算机的人造语言，是进行程序设计的工具，因此也称为程序设计语言。

程序设计语言可以分为**机器语言**、**汇编语言**、**高级语言**。高级语言种类繁多（据统计有上千种），曾经引起广泛关注和使用的高级语言有 FORTRAN、BASIC、Pascal 和 C 等命令式语言（或称过程式语言）；有 LISP、PROLOG 等陈述式语言；还有当前流行的面向对象的程序设计语言，例如 C++、Java、Visual C++、Visual Basic、Delphi、PowerBuilder 等。

计算机硬件能直接执行的是机器语言程序。汇编语言也称符号语言，用汇编语言编写的程序称汇编语言程序。计算机硬件不能识别和直接运行汇编语言程序，必须由“汇编程序”将其翻译成机器语言程序后才能识别和运行。同样，高级语言程序也不能被计算机硬件直接识别和执行，必须把高级语言程序翻译成机器语言程序才能执行。语言处理程序就是完成这个翻译过程的，按照处理方式的不同，可以分为解释型程序和编译型程序两大类。C 语言采

用编译程序，即把用 C 语言写的“源程序”编译成“目标程序”，再通过连接程序的连接，生成“可执行程序”才能运行。

简单的程序设计一般包含以下几个部分：

（1）确定数据结构，分析具体任务，确定输入数据和输出数据，确定数据的逻辑结构和存储结构；

（2）确定算法，根据确定的数据结构确定解决问题的方法，即完成任务的一步一步的步骤；

（3）编写程序，根据确定的数据结构和算法，使用选定的计算机语言编写程序代码，简称“编程”；

（4）调试程序，将编写完的程序输到计算机内存中，对程序进行测试并修正，直到程序符合任务要求；

（5）整理文档资料，根据数据结构和程序整理编写相关的文档资料。

3）算法

计算机解决问题所依据的步骤称为计算机算法，或简称算法。一个算法应具备以下五个基本特征。

（1）确定性。算法中每个操作步骤都应当是明确的，而不应是含糊的、模棱两可的。在计算机算法中最忌讳的是歧义性，所谓“歧义性”是指可以被理解为两种或多种可能的含义。因为计算机至今还没有主动思维的能力，如果给定的条件不确定，计算机就无法执行。

（2）有效性。算法中的每一个步骤都应当有效地执行，并得到确定的结果。例如当 b=0 时，a/b 是不能被有效执行的。

（3）有穷性。有穷性是指一个算法的操作步骤必须是有限的、合理的，即在合理的范围之内结束算法。例如求整数累加和的算法，由于整数本身是个无限集合，如果不限定其范围，会导致求解步骤是无限的。

（4）有零个或多个输入。执行算法时需要从外界获得必要信息的操作称为输入，输入数据的个数根据算法确定。例如计算 1～100 累加和的算法不需要输入，计算 n！的算法需要输入 n 的值。

（5）有一个或多个输出。执行算法得到的结果就是算法的输出，没有输出的算法是没有意义的。最常见的输出形式是屏幕显示或打印机输出，但并非唯一的形式。执行算法的目的就是为了求解，“解”就是输出。



2．算法的图形表示法

在程序设计过程中往往采用图形化设计方法进行概要设计。算法有多种表示方法，最常用的有流程图表示法和 N-S 图表示法。

（1）用流程图表示算法。流程图是用一组框图符号表示各种操作，也称框图。用流程图表示算法直观形象，易于理解。美国国家标准化协会 ANSI（American National Standard Institute）规定的一些常用流程图符号，已为各国程序工作者普遍采用，如图 1-1 所示。

结构体程序设计有三种基本结构：顺序结构、选择结构和循环结构。三种结构的流程图如图 1-2 所示。

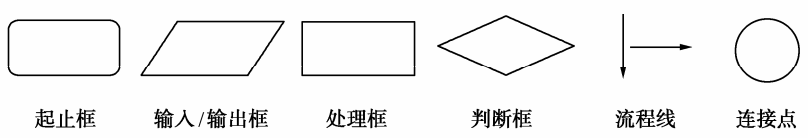


图 1-1　传统流程图的基本符号

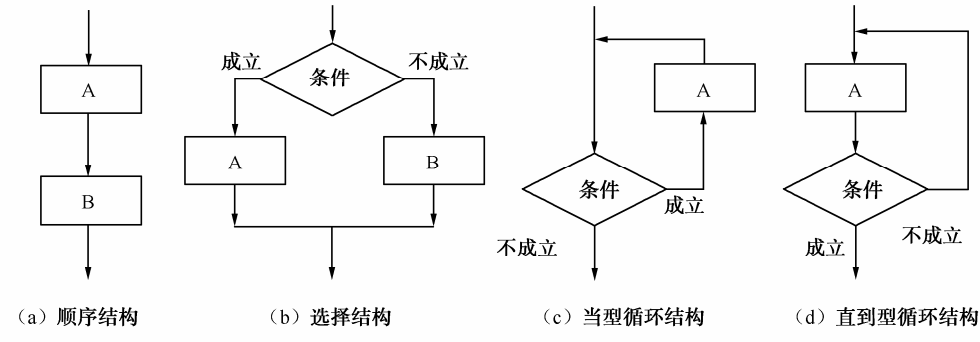


图 1-2　结构化的三种流程图

（2）用 N-S 图表示算法。N-S 图是美国学者 I.Nassi 和 B.Shneiderman 提出的一种新的流程图形式（N 和 S 是两位学者的英文姓名的首字母）。在 N-S 图中完全去掉了流程线，全部算法写在一个矩形框内，在该框内还可以包含其他的从属于它的框，即由一些基本框组成一个大框。N-S 图用图 1-3 所示的符号表示三种基本结构。

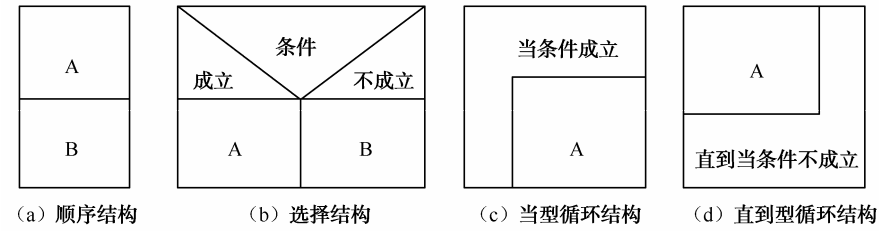


图 1-3　三种基本结构的 N-S 图符号



3．结构化程序设计概要

（1）一个结构化程序应符合的几个标准。

① 程序仅由顺序结构、选择结构和循环结构三种基本结构组成，基本结构可以嵌套。

② 每种基本结构都只有一个入口和一个出口，即一端进，一端出。这样的结构置于其他结构之间时，程序的执行顺序必然是从前一个结构的出口到本结构的入口，经本结构内部的操作，到达本结构的唯一出口。

③ 程序中没有死循环（不能结束的循环叫死循环）和死语句（程序中永远执行不到的语句叫死语句）。

（2）结构化程序设计方法遵循的原则。结构化程序设计方法遵循的原则是：自顶向下，逐步求精；模块化设计；结构化编程。下面分别说明。

① 自顶向下，逐步求精。把一个较大的复杂问题分解成若干相对独立而又简单的小问题，只要解决了这些小问题，整个问题也就解决了。

② 模块化设计。模块化程序设计早在低级语言时期就已经出现，但却在结构化程序设计的发展中得到充实、提高和完善。因此，它也是结构化程序设计的组成部分。

一般而言，模块化设计是把复杂的算法或程序，分解成若干相对独立、功能单一，甚至可供其他程序调用的模块。

③ 结构化编程。所谓结构化编程是指利用高级语言提供的相关语句实现三种基本结构，每个基本结构具有唯一的出口和入口，整个程序由三种基本结构组成，程序中不用 goto 之类的语句。goto 语句也称转移语句，用它可以改变程序中语句的执行次序。

二、C语言简介

【思考题 1-1】 北京奥运会到了，请你在屏幕上显示出“Welcome to Beijing!”的字样。



程序分析

对于初学者来说，要完成这样的一个程序，首先要进入到 Turbo C 2.0 开发界面，要了解 C 语言的开发环境，在下面的“小讲堂”中我们会对 turbo C 2.0 的开发环境有详细的介绍。其次要了解 C 语言源程序在组成结构上的特点，虽然有关内容还未介绍，但可从这个思考题中了解到组成一个 C 源程序的基本部分和书写格式。



编写程序代码

```
#include  <stdio.h>                    /*将库文件 stdio.h 包含到该文件中*/
main()                                 /* 主函数名 */
{                                      /* main 函数体开始 */
   printf("Welcome to Beijing!\n");    /* 在屏幕上显示一个字符串*/
}                                      /* main 函数体结束 */
```



调试运行程序

程序运行结果如下：

```
Welcome to Beijing!
```



4．C语言发展概述

C 语言是在 1972 至 1973 年间由美国的贝尔实验室的 D.M.Ritchie 和 K.Thompson 以及英国剑桥大学的 M.Richards 等为描述和实现 UNIX 操作系统而设计的。

最初的 C 语言是附属于 UNIX 操作系统环境的，而它的产生却可以更好地描述 UNIX 操作系统。时至今日，C 语言已独立于 UNIX 操作系统，成为微型、小型、中型、大型和超大型（巨型）计算机通用的一种程序设计语言。

随着 C 语言的不断发展、应用和普及，目前，C 语言已经能够在多种操作系统下运行，实用的 C 语言编译系统种类繁多，如 Microsoft C、Turbo C 等。



5．C语言程序的特点

（1）一个C语言源程序可以由一个或多个源文件组成。

（2）每个源文件可由一个或多个函数组成。

（3）一个源程序不论由多少个文件组成，都有一个且只能有一个main函数，即主函数。

（4）C程序由注释、编译预处理和程序主体组成。

（5）不管主函数在程序的什么位置，C程序都是从主函数开始执行的。

（6）函数体是左、右花括号{、}括起来的部分。

（7）每个语句和数据定义的最后必须有一个分号;。

（8）C语言书写格式自由，一行内可以写一个语句，也可以写多个语句。

（9）一个变量必须在声明后才能使用。

（10）C语言本身没有输入/输出语句。

（11）#include语句是编译预处理的文件包含语句，其作用是将由双引号或尖括号括起来的文件内容读入该语句位置处。



6．C语言的特点

（1）语言简洁、紧凑，使用方便、灵活。C语言只有32个关键字，程序书写形式自由。

（2）运算符丰富。

（3）数据结构丰富，具有现代化语言的各种数据结构。

（4）具有结构化的控制语句。

（5）语法限制不太严格，程序设计自由度大。

（6）C语言允许直接访问物理地址，能进行位（bit）操作，能实现汇编语言的大部分功能，可以直接对硬件进行操作。

（7）生成目标代码质量高，程序执行效率高。

（8）用C语言写的程序可移植性好（与汇编语言相比）。



7．C语言的书写规则

从书写清晰，便于阅读、理解和维护的角度出发，在书写程序时应遵循以下规则。

（1）一个说明或一个语句占一行。

（2）用{}括起来的部分，通常表示了程序的某一层次结构。{}一般与该结构语句的第一个字母对齐，并单独占一行。

（3）低一层次的语句或说明可以比高一层次的语句或说明缩进若干字符后书写（一般缩进2个英文字符），以便看起来更加清晰，增加程序的可读性。

在编程时应力求遵循这些规则，以养成良好的编程风格。

小讲堂

8．C程序的几个概念

1）源程序

用高级语言或汇编语言编写的程序称为源程序。源程序不能直接在计算机上执行，需要用编译程序将源程序翻译为二进制形式的代码。C 语言源程序的扩展名为“.c”。

2）目标程序

源程序经过编译程序翻译所得到的二进制代码称为目标程序，目标程序的扩展名为“.obj”。目标代码尽管已经是机器指令，但是还不能运行，因为目标程序还没有解决函数调用问题，需要将各个目标程序与库函数连接，才能形成完整的可执行程序。

3）可执行程序

目标程序与库函数连接，形成完整的可在操作系统下独立执行的程序称为可执行程序。可执行程序的扩展名为“.exe”（在 DOS/Windows 环境下）。

用户在编辑完 C 语言源程序（*.c）后，可以通过编译将源程序生成二进制的目标文件（*.obj）文件，然后再将目标文件连接生成可执行文件（*.exe）。如表 1-1 所示是源程序、目标程序和可执行程序三者之间的对照关系表。

可执行程序的运行结果是否正确需要经过验证，如果结果不正确则需要进行调试。调试程序往往比编写程序更困难、更费时间。如图 1-4 所示为 C 程序编辑、编译、连接和运行的全过程。

表 1-1　源程序、目标程序和可执行程序之间的关系

	源　程　序	目 标 程 序	可执行程序
内容	程序设计语言	机器语言	机器语言
可执行	不可以	不可以	可以
文件名后缀	.c	.obj	.exe

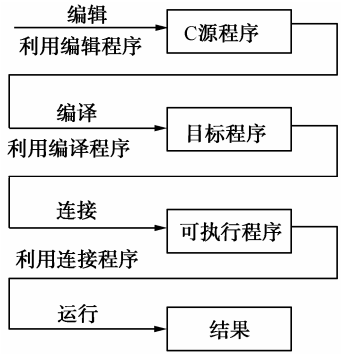


图 1-4　C 程序编辑、编译、连接、运行全过程

小讲堂

9．Turbo C 2.0 安装配置

1）解压缩 Turbo C 2.0

现在大多数的 Turbo C 2.0 都不需要安装，只要将其解压缩后即可使用。假设将其解压缩到 X 盘（可以是 C、D 等各盘）根目录下，其目录为 TC（即为 X:\TC）。

2）建立 Output 目录

Output 目录用来存放使用 TC 编写的执行文件和编译过程中使用的临时文件，虽然该目录不一定要有，但为了使用户今后可能编写的大量应用程序便于管理，强烈建议建立此目录。建立方法如下：

使用“资源管理器”或“我的电脑”打开 TC 所在目录，右键单击空白空间，在弹出的菜单中选择“新建”→“文件夹”命令，再将文件夹名称改为 Output。当然可以用任何名称的目录作为 Output 目录，但请切记要在后面的几步中作相应的变动才能使设置生效。

3）设置 TC 环境变量

首先运行 TC 目录下的 TC.exe，这是 TC2.0 编译环境的主程序，按 F10 键打开主菜单，选择“Options”→“Directories”命令，所要设定各选项对应意义如下：

```
Include directories：Include 头文件所在目录
Library directories：Lib 库文件所在目录
Output directory：临时文件和最终 EXE 文件输出目录
Turbo C directory：TC 所在目录
```

按照前边的目录结构，这里应如下设置：

```
Include directories：X:\TC\INCLUDE
Library directories：X:\TC\LIB
Output directory：X:\TC\OUTPUT
Turbo C directory：X:\TC
```

如果将 TC 解压缩到其他目录下的话，请相应修改设置（注意 X 为 TC 所在的盘符）。

4）保存设置

为了省去每次启动 TC 后都进行设置，在完成以上步骤之后还要将设置结果存盘，具体的操作是选择菜单“Options”→“Save options”。此时会询问所要存储的环境文件的名称，如果是个人使用可以按 Y 键直接选择覆盖默认文件；如果是多人共用同一 TC 可以通过每人存储设置，由不同 Output 路径的环境文件将各自的执行文件分开，以便于管理，每次使用时只要导入自己的环境文件即可。

至此，从安装到设置的一系列工作全部完成，TC 已经可以正常工作了，如果程序仍然无法编译运行，就可能是在编程过程中出现了语法问题，请仔细审查源码。最后要提醒的是程序的源代码极其重要，最好将它们存储到专门的目录，不要随便存在 TC 目录下，否则时间一长将很难再找到。



10．Turbo C 2.0 中的汉字显示

在 Turbo C 2.0 中，如果想输入和显示汉字，需要使用 UCDOS 汉字系统。但现在的 Windows 操作系统不支持 UCDOS 汉字系统的 16 位显示。下面介绍一种能在 Windows XP 系统环境下，在 Turbo C 2.0 中使用 UCDOS 的汉字系统的方法。

带 UCDOS 汉字系统的 Turbo C 2.0 启动方法如下：

首先启动命令提示符（MS-DOS 状态），方法是选择“开始”→“程序”→“附件”→“命令提示符”。（注意：以下命令中不带下画线文字为 DOS 的系统提示符，带下画线文字为用户输入的命令信息，“↙”表示回车键。假设 TC 和 UCDOS 文件夹都在 C 盘根目录内。）

```
C:\Documents and Settings>cd\↙          （将目录切换到 C 盘根目录下）
C:\>cd ucdos↙                           （进入 UCDOS 子目录）
C:\UCDOS>rd16↙                          （以下四条命令为启动 UCDOS 的命令）
C:\UCDOS>knl↙
C:\UCDOS>py↙
C:\UCDOS>rdfnt↙
C:\UCDOS>wb↙                            （五笔输入法，如不用五笔可省略此步）
```

```
C:\UCDOS>cd\↙                    （退出 UCDOS 子目录，回到 C 盘根目录下）
C:\>cd  tc↙                      （进入 TC 子目录）
C:\TC>tc↙                        （运行 TC.EXE 文件，启动 Turbo C 2.0）
```

这时就可以进入 Turbo C 2.0 的编辑界面，编辑、调试运行 C 语言源程序了。

如果不喜欢使用 UCDOS 的汉字系统，也可以使用其他 C 语言编辑器，如 Visual C++6.0 或 Turbo C 2.0 的汉化版。

Turbo C 2.0 的汉化版使用的汉化软件是 CCDOS97，安装后可以双击其快捷方式直接进入 Turbo C 2.0 中文界面，也可以直接使用各种汉字输入法，完全能够满足用户编辑 C 语言源程序中汉字输入与显示的需要。

用 Visual C++6.0 也可以运行 C 的源程序。只是 Turbo C 2.0 中的图形库函数在 Visual C++6.0 中不兼容，所以带图形库函数的程序不能在 Visual C++6.0 中运行，但不包含图形库函数的 C 源程序可以在 Visual C++6.0 中运行。



11. Turbo C 2.0 使用方法简介

很多上机实习和考试的环境都是使用 Borland Turbo C 2.0 这个版本。该系统是 DOS 操作系统支持下的软件，在 Windows 98、Windows 2000 及 Windows XP 环境下，可以在 DOS 窗口下运行。

Turbo C 2.0 默认的路径是 C 盘根目录，但一般来讲学校机房是在 C 盘根目录下建立一个 TC 子目录来安装 Turbo C 2.0 系统的。TC 目录下还建立有两个子目录 LIB 和 INCLUDE，LIB 子目录中存放库文件，INCLUDE 子目录中存放所有头文件。

进入 Turbo C 2.0 集成开发环境中后，屏幕上显示如图 1-5 所示。其中最上面一行为 Turbo C 2.0 主菜单，中间窗口为编辑区，再下面是信息窗口，最下面一行为参考行。这四个窗口构成了 Turbo C 2.0 的主界面，以后的编程、编译、调试及运行都将在这个主界面中进行。除 Edit 外，其他各项均有子菜单，只要用 Alt 加上某项中第一个字母，就可进入该项的子菜单中。

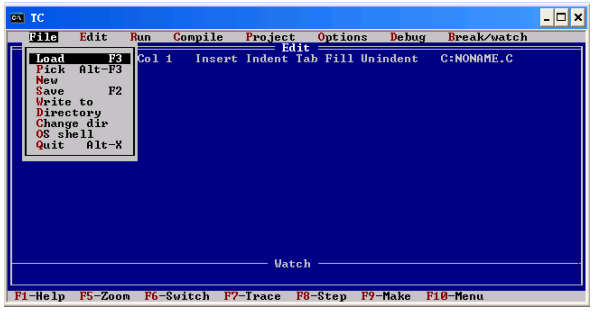


图 1-5　进入 Turbo C 2.0 初始运行界面

1）Turbo C 2.0 常用快捷键

（1）常用的 Turbo C 2.0 快捷键：

Alt+F——文件菜单

- Load（F3）——打开文件
- Pick——从目录中选择文件
- Save（F2）——保存
- Write to——另存为

- Quit——退出

Alt+E——编辑（显示光标。若调试有错不显示光标时，可将光标重新显示在编辑区）
Alt+R——运行菜单

- Run（或 Ctrl+F9）——运行程序
- User Screen（或 Alt+F5）——用户界面，查看运行结果

Alt+O——选项菜单

- Directories——设置路径（具体步骤见 Turbo C 2.0 的安装配置部分）

Alt+F9（等价于 Compile）——编译
Alt+回车键——全屏/半屏切换（将英文版的 TC 以程序的方式显示在桌面上）
F9——编译系统查错
F10——调用主菜单

（2）带 UCDOS 汉字系统的 Turbo C 2.0 中，如果想更改汉字输入法，快捷键如下：

Alt+F1——区位　　Alt+F2——全拼
Alt+F3——双拼　　Alt+F5——五笔
Alt+F6——英文　　单次按右 shift 键　　可显示或隐藏中文输入法

（3）全/半角方式切换键：

Ctrl+F9　　（注意在中文输入条显示的时候有效）

（4）块操作：

Ctrl+KB——块开始标记　　Ctrl+KK——块结束标记
Ctrl+KC——块拷贝　　Ctrl+KV——块移动
Ctrl+KY——块删除　　Ctrl+KH——块隐藏

2）汉化版的 Turbo C 2.0 常用功能键

（1）切换输入法：

Alt+F1——国标区位码输入法　　Alt+F2——智能拼音输入法
Alt+F4——王码五笔字型输入法　　Alt+F9——特殊符号输入法

注意　当连续切换输入法时，要先松开 Alt 键再进行下一次切换。

（2）中/英文键盘切换键：Ctrl+空格键。设置为 Ctrl+空格键以配合 Windows 环境习惯，用户可修改 CCDOS.INI 配置文件，修改为其他切换方式。

（3）全/半角方式切换键：Ctrl+F9。

注意　在运行 Turbo 系列程序时，由于与程序运行热键相同，则应先按 Ctrl+空格键关闭 CCDOS 输入工具条，然后再使用此热键。

（4）CCDOS 系统设置键：Ctrl+F5。进入 CCDOS 系统设置功能，可在 DOS 中退出 CCDOS 中文环境。

小讲堂

12．如何在TC中调试程序

程序调试（debug）是程序设计中的一项基本技能，不会调试程序，算不上会设计程序。下面就以 TC 2.0 为例说明如何设置断点、单步跟踪、监视变量等调试方法，该方法在 TC 3.0 中同样适用。

在 TC 的菜单中，每个主菜单项都有一个红色的打头字母，表示该菜单的快捷键是 Alt+该字母，比如 File 菜单的快捷键是 Alt+F。

首先了解与程序调试相关的菜单项和快捷键。

（1）设置断点（Ctrl+F8 快捷键，菜单“Break/watch”→“Add watch”）。断点就是要

求程序暂停的一行，把光标移到这行，按 Ctrl+F8 快捷键，出现红色横条的行就是断点所在行，一个程序中可以设置多个断点。当再次按 Ctrl+F8 时，该断点被取消。

按 Ctrl+F9 运行程序时，在断点处暂停，以便观察。如果在循环中设置断点，循环一次暂停一次。

（2）单步运行：（F7 快捷键，菜单“Run”→“Trace into”）。按一次 F7，程序执行一步，然后暂停。一般先运行到设置断点处，再从断点处开始单步运行。

注意 当有函数的调用时，F7 要跟踪到函数的内部；F8 不跟踪到函数的内部，只把函数当一句话。

（3）即时计算表达式的值（Ctrl+F4 快捷键，菜单“Debug”→“Evaluate”）。在程序暂停运行时，可以在对话框中输入感兴趣的表达式，看得到的值与预期的是否一致。

（4）全程监视表达式的值（Ctrl+F7 快捷键，菜单“Break/watch”→“Add watch”）。先按 F5 快捷键打开监视（Watch）窗口，再按 Ctrl+F7 快捷键，输入要一直监视的表达式，可以在程序单步运行的过程中对每一步的结果进行监视。

如果要清除监视的表达式，选择“Break/watch”→“Clear all breakpoints”命令即可。

小讲堂

13．在Visual C++6.0 中调试C源程序步骤

（1）打开 Visual C++6.0 后，单击“文件”菜单中的“新建”，弹出“新建”对话框，如图 1-6 所示。

（2）选择“工程”标签，单击“Win32 Console Application”建立一个 DOS 项目。单击对话框右侧“位置”后的... 按钮，在弹出对话框中选择该工程的位置。在“工程名称”中输入该工程名，如“example”，单击“确定”按钮。

（3）弹出一个新的对话框，选择要创建什么类型的控制台。这里我们选择第一个选项“一个空工程”，单击“完成”按钮。

（4）此时又弹出一个对话框，提示该工程的信息（存放位置等），单击“确定”按钮，这时就建立了一个空的工程，名为“example”。

（5）再建立一个 C 语言源程序文件。选择“文件”菜单中的“新建”命令，在弹出的“新建”对话框中选择“文件”标签，在右侧“文件名”中输入文件名，如“ex1.c”，如图 1-7 所示。

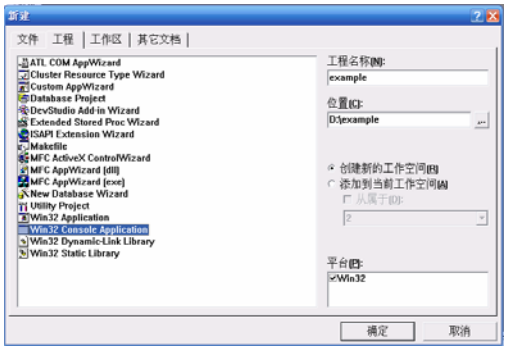


图 1-6 “新建”对话框“工程”标签

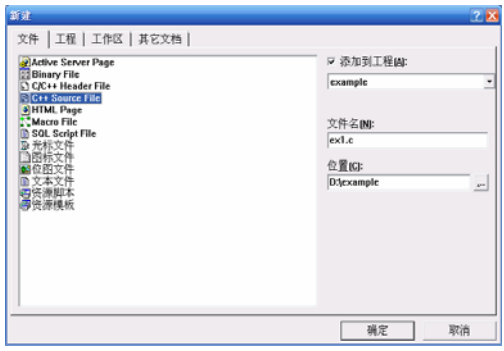


图 1-7 “新建”对话框“文件”标签

这样就在工程“example”中建立了一个.c 的源程序。在程序编辑区输入程序代码后，单击工具栏中的“Complile（Ctrl+F7）”按钮进行编译，再单击“Build（F7）”按钮建立该应用程序，单击“BuildExecute（Ctrl+F5）”按钮即可运行，并显示运行结果。

练一练

【练习 1-1】 把 Turbo C 2.0 文件夹放在 C 盘根目录下，看一看运行【思考题 1-1】是什么结果？如果把 Turbo C 2.0 文件夹存放在 D 盘根目录下，又会出现什么结果？

解：当 Turbo C 2.0 不是安装在 C 盘根目录下时，会出现错误，这是因为 C 语言默认的安装路径是在 C 盘根目录下（“c:\tc”），如果不是在这个路径下则找不到系统相关的文件，运行出错。可以将 Turbo C 2.0 系统的目录改为用户安装的目录，如 TC 安装在 D 盘时，可将 TC 的系统默认路径改为“d:\tc”。方法是：按 Alt+O 快捷键（Options 菜单）后选择 Directories 选项，在弹出的窗口中将所有路径中的“c:\tc\”都改为当前 Turbo C 2.0 的安装路径（如“d:\tc\”），再次运行程序就没有错误了。

【练习 1-2】 如果运行结果如下图所示，则该程序如何编写？



解：按照本讲【思考题 1-1】所举例程，可以写出相应程序如下：

```
#include  <stdio.h>
 main()
{  printf("one world\n");
   printf("one dream\n");
}
```

想一想

如果计算一个正方形的面积，这样的程序该如何写，需要哪些组成元素？

本章小结

本讲主要介绍了 C 语言的基础知识，主要包括程序设计的概念、算法的特点、程序设计算法的图形表示法（包括流程图表示法和 N-S 图表示法两种）、C 语言的特点、Turbo C 2.0 的集成开发环境、TC 的上机步骤和常用热键等。

课后习题一

一、选择题

1．以下_____不是 C 语言的特点。

A）生成目标代码质量高　　B）语法限制严格

C）数据类型丰富　　D）控制流程结构化

2．以下_____不是算法的特征之一。

A）有穷性　　B）确定性　　C）有效性　　D）复杂性

3．以下_____不是二进制代码文件。

A）标准库文件　　B）目标文件　　C）源程序文件　　D）可执行文件

4．以下_____是流程图的判断框。



5．以下_____是流程图的起止框。



6．以下_____是流程图的处理框。



7．以下_____是流程图的输入/输出框。



8．C 语言程序的基本单位是_____。

A）程序行　　B）语句　　C）函数　　D）字符

9．C 语言可执行程序的开始执行点是_____。

A）程序中第一条可执行语句　　B）程序中第一个函数

C）程序中的 main 函数　　D）包含文件中的第一个函数

10．一个完整的 C 源程序是_____。

A）要由一个主函数（或）一个以上的非主函数构成

B）由一个且仅由一个主函数和零个以上（含零）的非主函数构成

C）要由一个主函数和一个以上的非主函数构成

D）由一个且只有一个主函数或多个非主函数构成

二、填空题

1．函数体以符号_____开始，以符号_____结束。

2．一个完整的 C 程序至少要有一个_____函数。

3．程序设计语言可以分为_____、_____、_____。

4．结构化程序设计方法遵循的原则是：_____、_____、_____。

5．C 程序是以_____为基本单位，整个程序由_____组成。

6．C 源程序文件的扩展名是_____，C 目标文件的扩展名是_____，C 可执行文件的扩展名是_____。

7．程序连接过程是将目标程序、_____或其他目标程序连接装配成可执行文件。

第二章　数据描述与基本操作

数据和操作是构成程序的两个要素。计算机处理的对象是数据，而数据是以某种特定形式存在的。为了能准确、方便地使用数据描述生活中的各种信息，C 语言将数据划分为不同的数据类型。C 语言提供了丰富的运算符，构成多种表达式，能实现多种基本操作。在本章中，主要介绍基本数据类型、运算符和表达式，以及基本的输入/输出函数的使用方法。

第二讲　基本数据类型、变量与常量



学习目标

- 了解基本标识符;
- 掌握 C 语言的变量;
- 掌握 C 语言的常量。

【思考题 2-1】　已知一个圆的半径，求这个圆的周长和面积。



程序分析

如果我们用 r 代表圆的半径，用 l 代表圆的周长，用 s 代表圆的面积，用 PI 代表圆周率π，那么圆的周长公式是 l=2πr，圆的面积公式是 $s=\pi r^2$。在这两个公式里，圆周率 PI 是固定不变的，而半径是可变的，则圆的周长与圆的面积也是可变的。那么，这些元素如何在 C 语言里进行说明并表示出来？这就是这一讲要重点讲述的内容。



编写程序代码

```
#include  "stdio.h"            /*文件包含命令*/
#define  PI  3.14              /*定义符号常量圆周率π*/
main()
{  int r;                      /*定义圆的半径 r*/
   float  area,length;         /*定义圆的面积 s 和周长 l，为单精度浮点型*/
   r=3;                        /*为半径变量 r 设置初始值 3*/
   length=2*PI*r;              /*计算圆周长，并把结果赋值给 l*/
   area=PI*r*r;                /*计算圆的面积，并把结果赋值给 s*/
   printf("r=%d\nlength=%f\narea=%f\n",r,length,area);  /*输出运算结果*/
}
```



调试运行程序

按 Ctrl+F9 快捷键，对程序进行编译，然后按 Alt+F5 快捷键查看结果，程序运行结果如下：



小讲堂

1．标识符

通过上面的程序，我们先想一想 include，define，main，PI，float，r，area，length，printf 等都代表了什么？在程序中起到了什么作用？要解决这个问题，就先来了解一下什么是标识符。

所谓标识符，是指用来标识程序中用到的变量、函数、类型、数组、文件以及符号常量等的有效字符序列。简言之，标识符就是一个名字。C 语言中的标识符可以分为三类：即关键字、预定义标识符和用户自定义标识符。

1）关键字

关键字又称保留字，是 C 语言规定的具有特定意义的标识符，它已被 Turbo C 2.0 本身使用，不能做其他用途使用，每个关键字都有固定的含义。C 语言的关键字分为以下四类。

（1）标识数据类型的关键字（14 个）。int，long，short，char，float，double，signed，unsigned，struct，union，enum，void，volatile，const。

（2）标识存储类型的关键字（5 个）。auto，static，register，extern，typedef。

（3）标识流程控制的关键字（12）个。goto，return，break，continue，if，else，while，do，for，switch，case，default。

（4）标识运算符的关键字（1 个）。sizeof。

例如【思考题 2-1】程序中的“float”就是数据类型的关键字，它表示单精度浮点型数据。

2）预定义标识符

预定义标识符是一类具有特殊含义的标识符，用于标识库函数名和编译预处理命令。系统允许用户把这些标识符另做他用，但这将使这些标识符失去系统规定的原意，为了避免误解，建议不要将这些预定义标识符另做他用。

C 语言中常见的有以下几种。

（1）编译预处理命令。define，ifdef，ifndef，endif，include，line，if，else 等。

例如【思考题 2-1】程序中的“include”和“define”就是两个预定义标识符，要在其前面加“#”号，一般写在主函数 main 的上面。

（2）标准库函数。包括数学函数 sqrt，fabs，sin，pow 等，还有输入/输出函数 scanf，printf，getchar，putchar，gets，puts 等。

例如程序中的“printf”就是一个输出函数，通过这个函数可以把结果在显示器上显示出来。

3）用户自定义标识符

用户自定义标识符是程序员根据自己的需要定义的用于标识变量、函数、数组等的一类标识符。用户在定义标识符时应符合 C 语言标识符的命名规则。

在 C 语言中，标识符的命名规则如下：

- 标识符只能由字母、数字和下画线三种字符组成；
- 第一个字符必须为字母或下画线。

例如【思考题 2-1】程序中的 PI、r、l、s 就是用户自己定义的标识符，只要符合 C 语言标识符的命名规则即为合法。

用户在定义标识符时，还应该注意以下四点：

（1）用户在定义标识符时不应采用与 C 语言中关键字同名的标识符；

（2）系统已经定义了一些预定义标识符，包括预处理命令名和 C 语言提供的库函数的名字，为了增强程序的可读性，建议用户不将其定义为标识符使用；

（3）C 语言是“大小写敏感”的语言，即将大写字母和小写字母作为不同的字符处理，C 语言中的关键字和预定义标识符全部以小写字母表示；

（4）不同的 C 语言版本对标识符的长度有不同的规定，许多系统通常取前 8 个字符，因此在定义标识符时最好不要超过 8 个字符。



2．基本数据类型

我们再读一遍【思考题 2-1】这个程序，程序中已给出 r 的值为 3，程序运行显示结果为：

```
l=18.840000
s=28.260000
```

这里 r 的值为 3，是整数，而 l、s 为小数的形式，也就是说，r 与 l、s 是不同的数据类型。那么 C 语言的数据类型是怎样规定的呢？

在 C 语言中，每个数据都属于唯一的一种数据类型，没有无类型的数据。C 语言的数据类型如图 2-1 所示。

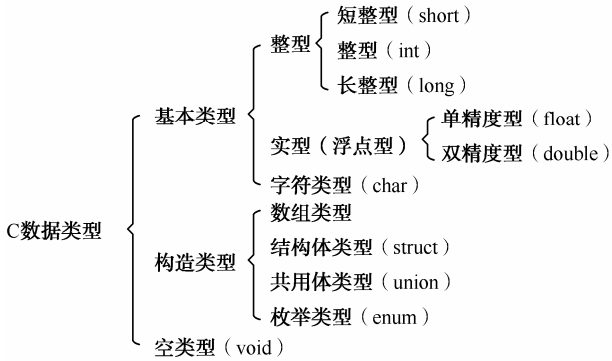


图 2-1　C 语言的数据类型

C 语言的基本类型有三种：字符型、整型和实型（浮点型）。C 语言的基本类型修饰符有四种：signed（有符号）、unsigned（无符号）、long（长型符）和 short（短型符），这些类型修饰符可以与字符型或整型数据配合使用。基本数据类型和取值范围如表 2-1 所示。

表 2-1　C 语言基本数据类型描述

类　　型	说　　明	内存单元个数	取 值 范 围
char	字符型	1（8 位）	−128～127　　即-2^7～(2^7-1)
unsigned char	无符号字符型	1（8 位）	0～255　　即 0～（2^8-1）
signed char	有符号字符型	1（8 位）	−128～127　　即-2^7～（2^7-1）
int	整型	2（16 位）	−32768～32767　　即-2^{15}～$(2^{15}-1)$

续表

类　型	说　明	内存单元个数	取 值 范 围
unsigned int	无符号整型	2（16 位）	0～65535　即 0～(2^{16}–1)
signed int	有符号整型	2（16 位）	–32768～32767　即–2^{15}～（2^{15}–1）
short int	短整型	2（16 位）	–32768～32767　即–2^{15}～（2^{15}–1）
unsigned short int	无符号短整型	2（16 位）	0～65535　即 0～(2^{16}–1)
signed short int	有符号短整型	2（16 位）	–32768～32767　即–2^{15}～（2^{15}–1）
long int	长整型	4（32 位）	–2147483648～2147483647　即–2^{31}～（2^{31}–1）
unsigned long int	无符号长整型	4（32 位）	0～4294967295　即 0～（2^{32}–1）
signed long int	有符号长整型	4（32 位）	–2147483648～2147483647　即–2^{31}～（2^{31}–1）
float	单精度实型	4（32 位）	–3.4E+38～3.4E+38
double	双精度实型	8（64 位）	–1.7E+308～1.7E+308

从表 2-1 可以看出，int 和 short int 是等价的，unsigned int 和 unsigned short int 等价。应注意的是，不同系统的规定可能不同。而本讲程序中的 r，s，l 则都属于 float 数据类型。



3．变量

所谓变量，是指在程序运行过程中其值可以改变的量。一个变量应该有一个名字，在内存中占据一定的存储单元。变量定义必须放在变量使用之前，一般放在函数体的开头部分。要区分变量名和变量值是两个不同的概念。例如整型变量 a 的值为 3，则变量 a 在内存中的存储形式如图 2-2 所示。

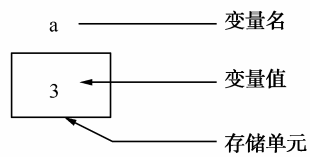


图 2-2　变量及其存储示意图

Turbo C 2.0 规定所有变量在使用前都必须加以说明。一条变量说明语句由数据类型和其后的一个或多个变量名组成。

1）变量定义的格式

```
存储类型符　数据类型符　变量名表;
```

说明：

（1）存储类型符用来说明变量的存储类型，存储类型可以是自动类型（auto）、寄存器类型（register）、静态类型（static）、外部类型（extern）。默认为自动类型（auto），如果对存储类型未做任何说明，则按默认的自动类型处理。

（2）数据类型符用来说明变量的数据类型，数据类型可以是 C 语言中任意一种基本数据类型或构造数据类型标识符。

（3）变量名表中可以只有一个变量，也可以有多个变量，如果有多个变量，变量之间用逗号隔开。

例如，【思考题 2-1】中的两行程序代码：

```
int  r;
float  s,l;
```

定义了一个整型变量 r 和两个单精度浮点型变量 s、l。这两行程序代码都没有声明存储类型，Turbo C 中就按 auto（自动类型）进行处理。“int”、“float”为数据类型标识符，是 C 语言的

关键字，用来说明变量的类型；“r”、“s”、“l”是变量名。

2）变量的赋值

定义变量后，在使用之前需要给定一个初始值。在 C 语言中，可以通过赋值运算符“=”给变量赋值。变量赋值语句的一般格式是：

```
变量名=表达式;
```

变量的赋值，一般有以下两种情况。

（1）先定义变量，后赋值。如【思考题 2-1】程序开头的部分代码：

```
int  r;
…
r=3;
```

（2）变量的初始化。在定义变量的同时为其赋值，称为变量的初始化。定义的变量可以全部初始化，也可以部分初始化。对于上面的程序段也可以写成：

```
int  r=3;
```

即定义了整型变量 r 的同时，对其赋初值为 3。

3）赋值的注意事项

在给变量赋值时，应注意以下几个问题。

（1）变量在某一时刻只有一个确定的值，变量获得新值后，其原值将不再存在。例如：

```
int  r;
…
r=3;
r=2;
```

该程序执行后，变量 r 的值是 2，而不是 3。

（2）定义多个同类型变量时，如果给所有变量赋同一个值，只能逐个处理。如有三个整型变量 x，y，z，且初值均为 10，可以写成下面的形式：

```
int  x=10,y=10,z=10;
```

（3）如果变量的类型与所赋数据的类型不一致，所赋数据将被转换成与变量相同的类型。例如，下面的定义是合法的：

```
int  x=10.5;
long  y=99;
```

该程序执行后，变量 x 的值是整数 10（只将整数部分赋给变量 x），变量 y 的值是长整数 99。

小讲堂

4．常量

常量又称常数，是指在程序运行过程中其值不能改变的量。C 语言中的常量又分字面常量和符号常量。字面常量又称直接常量，就是常说的常数。字面常量可以分为不同的类型，有整型常量、实型常量、字符型常量与字符串常量。

1）整型常量

整型常量又称整数，在 C 语言中，整数可以用三种数制来表示。

（1）十进制整常数：十进制整常数没有前缀，其数码为 0～9。以下各数是合法的十进制整常数：237、-568、65535、1627。

以下各数不是合法的十进制整常数：023（不能有前导 0），23D（含有非十进制数码）。

在程序中是根据前缀来区分各种进制数的，因此在书写常数时不要把前缀弄错造成结果不正确。

（2）八进制整常数：八进制整常数必须以 0 开头，即以 0 作为八进制数的前缀。数码取值为 0～7。八进制数通常是无符号数。以下各数是合法的八进制整常数：

015（十进制数为 13），0101（十进制数为 65），0177777（十进制数为 65535）。

以下各数不是合法的八进制整常数：

256（无前缀 0），03A2（包含了非八进制数码），－0127（出现了负号）。

（3）十六进制整常数：十六进制整常数的前缀为 0X 或 0x，其数码取值为 0～9、A～F 或 a～f。以下各数是合法的十六进制整常数：

0X2A（十进制数为 42），0XA0（十进制数为 160），0XFFFF（十进制数为 65535）。

以下各数不是合法的十六进制整常数：

5A（无前缀 0X），0X3H（含有非十六进制数码）。

（4）整型常数的后缀：在 16 位字长的机器上，基本整型的长度也为 16 位，因此表示数的范围也是有限定的。十进制无符号整常数的范围为 0～65535，有符号数为−32768～+32767；八进制无符号数的表示范围为 0～0177777；十六进制无符号数的表示范围为 0X0～0XFFFF 或 0x0～0xFFFF。如果使用的数超过了上述范围，就必须用长整型数来表示。长整型数是用后缀“L”或“l”来表示的。

例如，十进制长整常数：

158L（十进制数为 158），358000L（十进制数为 358000）。

八进制长整常数：

012L（十进制数为 10），077L（十进制数为 63），0200000L（十进制数为 65536）。

十六进制长整常数：

0X15L（十进制数为 21），0XA5L（十进制数为 165），0X10000L（十进制数为 65536）。

长整数 158L 和基本整常数 158 在数值上并无区别。但对 158L，因为是长整型量，C 编译系统将为它分配 4 个字节的存储空间；而对 158，因为是基本整型，只为其分配 2 个字节的存储空间。因此在运算和输出格式上要予以注意，避免出错。

无符号数也可用后缀表示，整型常数的无符号数的后缀为“U”或“u”。例如：358u，0x38Au，235Lu 均为无符号数。

前缀和后缀可同时使用以表示各种类型的数，如 0XA5Lu 表示十六进制无符号长整数 A5，其十进制数为 165。

2）实型（浮点型）常量

实型也称为浮点型，实型常量也称为实数或者浮点数。在 C 语言中，实数只采用十进制。它有两种形式：十进制小数形式和指数形式。

（1）十进制小数形式。小数形式是由数码 0～9 和小数点组成的（注意：必须有小数点）。例如：6.789，.789，6.，0.0 都是合法的十进制小数。

（2）指数形式。指数形式又称科学计数法。由十进制小数加上阶码标志“e”或“E”以及阶码（只能为整数，可以带符号）组成。其一般形式为：

```
a E n
```

其中 a 为十进制数，n 为十进制整数，其值为 $a*10^n$。

指数形式的表示法有两点需要注意：e 或 E 前、后必须有数字，e 或 E 后的数字必须是整数。

例如，以下为合法的实数：2.1E5（等于 $2.1*10^5$），3.7E−2（等于 $3.7*10^{-2}$），0.5E7（等于 $0.5*10^7$），−2.8E−2（等于$-2.8*10^{-2}$）。以下不是合法的实数：345（无小数点），E7（阶码标志 E 之前无数字），−5（无阶码标志），53.−E3（负号位置不对），2.7E（无阶码）。

标准 C 允许浮点数使用后缀，后缀为“f”或“F”即表示该数为浮点数，如，356f 和 356.是等价的。

3）字符型常量

（1）字符常量的定义。字符常量是用**单**引号括起来的**一个**字符，如：'a'，'A'，'@'，'?'等。

（2）字符常量的特点。字符常量只能用单引号括起来，不能用双引号或其他括号。单引号只是字符与其他符号的分隔符，或者说是字符常量的定界符，不是字符常量的一部分，当输出一个字符常量时不输出此单引号。

字符常量只能是单个字符，不能是多个字符。

字符可以是字符集中的任意字符，但数字被定义为字符型之后就不能参与数值运算，如，'5'和 5 是不同的，'5'是字符常量，不能参与运算。

（3）字符型数据在内存中的存储形式及使用方法。每个字符变量被分配一个字节的内存空间，因此只能存放一个字符。字符值是以 ASCII 码的形式存放在变量的内存单元之中的。

如，x 的十进制 ASCII 码是 120，y 的十进制 ASCII 码是 121，对字符变量 a 和 b 赋值为字符'x'和'y'的语句为：

```
a='x';  b='y';
```

实际上是在 a，b 两个单元内存放 120 和 121 的二进制代码：

a:	0	1	1	1	1	0	0	0
b:	0	1	1	1	1	0	0	1

所以也可以把它们看成是整型量。C 语言允许对整型变量赋以字符值，也允许对字符变量赋以整型值。在输出时，允许把字符变量按整型量输出，也允许把整型变量按字符量输出。

整型常量在内存中占 2 个字节，字符常量在内存中占 1 个字节，当整型量按字符型量处理时，只有低 8 位字节参与处理。

转义字符是一种特殊的字符常量。转义字符以反斜线“\”开头，后跟一个或几个字符。转义字符主要用来表示那些用一般字符不便于表示的控制代码，例如“回车”、“换行”等。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符，如表 2-2 所示。

表 2-2　转义字符及其功能

字　　符	功　　能	ASCII 码
\0	表示字符串结束	0
\n	换行，将当前位置换到下一行行首	10
\t	横向跳格，即从当前位置跳到下一个输出区	9
\v	竖向跳格	11

续表

字　符	功　能	ASCII 码
\b	退格，将当前位置移到前一列	8
\r	回车，将当前位置移到本行行首	13
\f	换页，将当前位置移到下页开头	12
\a	响铃	7
\ '	单引号字符	39
\ "	双引号字符	34
\\	反斜杠字符	92
\ooo	1～3 位八进制数表示的 ASCII 码所代表的字符	0～255
\xhh	1～2 位十六进制数表示的 ASCII 码所代表的字符	0～255

表 2-2 中的最后两行是用 ASCII 码（八进制或十六进制）表示一个字符，如，'\101'或'\x41'表示 ASCII 码为十进制数 65 的字符“A”，其中'\101'中 101 是八进制数，转换成十进制数就是 65，而 x41 是十六进制数，转换成十进制数也是 65。

4）字符串常量

（1）字符串常量的定义。字符串常量是用一对双引号括起来的零个或多个字符序列，如："hello"，""，"abc"，"123"等。

字符串以双引号为定界符，但双引号并不属于字符串。要在字符串中插入双引号，要借助转义字符"\ "才行。字符串的长度等于字符串中包含的字符个数，如字符串"hello"的长度为 5 个字符。

（2）字符常量与字符串常量的区别。字符串常量和字符常量是不同的量，它们之间主要有以下区别：

① 字符常量由单引号括起来，字符串常量由双引号括起来。

② 字符常量只能是单个字符，字符串常量则可以含一个或多个字符。

③ 可以把一个字符常量赋予一个字符变量，但不能把一个字符串常量赋予一个字符变量。在 C 语言中没有相应的字符串变量，这与 BASIC 语言是不同的。但是可以用一个字符数组来存放一个字符串常量，在数组一章会予以介绍。

④ 字符常量占一个字节的内存空间。字符串常量占的内存字节数等于字符串中字节数加 1，增加的一个字节中存放字符'\0'（ASCII 码为 0），这是字符串结束的标志。

例如：字符串 "C program" 在内存中所占的字节形式为：

'C'		'p'	'r'	'o'	'g'	'r'	'a'	'm'	'\0'

又例如：字符常量'a'和字符串常量"a"虽然都只有一个字符，但在内存中的情况是不同的。

'a'在内存中占 1 个字节，可表示为：

'a'

"a"在内存中占 2 个字节，可表示为：

'a'	'\0'

5）符号常量

在 C 语言中，也可以用一个标识符来表示一个常量，称之为符号常量。符号常量在使用之前必须先定义，其一般形式为：

```
#define    标识符    常量
```

在本讲的开头的思考题中，圆周率π用 PI 表示，可写为：

```
#define  PI  3.14
```

其中“define”为关键字，前面加“#”，表示一条预处理命令（预处理命令都以“#”开头），称为宏定义命令（会在以后编译预处理的章节中详细讲解）。

该语句的功能是把标识符“PI”定义为其后的常量值“3.14”。一经定义，以后在程序中所有出现“PI”的地方均代之以该常量值“3.14”。符号常量的标识符是由用户自行定义的。如果需要把圆周和圆的面积计算得更加精确，只需要把π的取值再精确一些，如将π的值取3.1415926，这行代码则可以改写为：

```
#define  PI  3.1415926
```

符号常量的特点是：

- 习惯上符号常量的标识符用大写字母，变量标识符用小写字母，以示区别；
- 符号常量与变量不同，它的值在其作用域内不能改变，也不能再被赋值。

使用符号常量的好处是：含义清楚，并且能做到“一改全改”。

练一练

【练习 2-1】 请判断下列哪些用户自定义标识符是合法的？（　　）

A）a&b，1_xy，e5，a.b

B）exam，x1，int，define

C）ram，_mn，3ep，x*y

D）ch，x_3_1，z2，num

解：因为标识符只能由英文大、小写字母，数字和下画线三种字符组成，且开头字母只能由字母和下画线组成，所以只有 D 中的所有标识符是正确的。注意用户自定义标识符不能是系统的保留字（关键字），而 C 语言中的 int 和 define 是关键字，不能做标识符，也是错误的。

【练习 2-2】 下面哪个是合法的字符串常量？（　　）

A）'a'　　B）'\076'　　C）"Hello"　　D）Hello

解：因为字符常量是用一对单引号括起来的一个字符，字符串常量是用一对双引号括起来的零个或多个字符序列，所以符合题目的必须得有一对双引号，里面可以有 n（n≥0）个字符的序列。选择 C。而 A 和 B 都是字符常量，D 可以是一个标识符。

【练习 2-3】 下面哪个表示浮点数的科学计数表示法是正确的？（　　）

A）3e2.5　　B）1.25e　　C）3.45e−5　　D）e+8

解：因为科学计数法有两点要求：（1）e 或 E 前、后必须有数字，（2）e 或 E 后的数字必须是整数，所以答案 C 是正确的。A 错误在于 e 后的数字为小数，B 错误在于 e 后没有数字，D 错误在于 e 前面没有数字。

【练习 2-4】 下面的变量定义及初始化语句哪个是正确的？（　　）

A）int　a=3;b=5;　　B）int　a=3 b=5

C）int　a=3,b=5;　　D）int　a==3,b==5;

解：因为变量定义及初始化语句格式是：

```
类型符　变量名 1=初值 1，变量名 2=初值 2，…；
```

各变量之间用逗号分隔，语句结束标志为分号“;”，变量初始化的赋值运算符为“=”，所以正确答案为 C。

本讲小结

本讲主要介绍了基本标识符的定义及其分类情况，C 语言的数据类型及各代表符号，运算符的使用，变量的定义、初始化、赋值，常量的定义与应用。

想一想

在本讲中，我们学习了 C 语言的基本数据类型，知道了每一种数据类型都有一个存储空间，即内存单元个数。大家看一看下面的这个程序，能不能说出运行结果是什么，为什么？

```
main()
{  int  a,b=20000,c=30000;
   c=a+b;
   printf("c=%d",c);
}
```

第三讲　运算符与表达式、数据类型的转换



学习目标

- 掌握算术运算符与算术表达式;
- 掌握关系运算符与关系表达式;
- 掌握逻辑运算符与逻辑表达式;
- 掌握赋值运算符与赋值表达式;
- 掌握条件运算符与条件表达式;
- 了解各种其他运算符。

一、算术运算与赋值运算

【思考题 2-2】　有两个变量 x、y，x 的初始值为 9，y 的初始值为 4，分别求出 x 与 y 的和、差、积、商与余数。



程序分析

通过上一讲中介绍的变量的声明与初始化，可以对变量 x 与 y 进行声明与初始化，用变量 a、b、c、d、e 分别表示 x 与 y 的和、差、积、商与余数。



编写程序代码

```
main()
{  int  x=9,y=4,a,b,c,d,e;
   a=x+y;
   b=x-y;
   c=x*y;
   d=x/y;
   e=x%y;
   printf("x+y=%d\nx-y=%d\nx*y=%d\nx/y=%d\nx%%y=%d\n",a,b,c,d,e);
}
```



程序运行结果如下：

```
x+y=13
x-y=5
x*y=36
x/y=2
x%y=1
```



（1）x/y 的结果为 2，而不是 2.25，这是因为“/”号在 C 语言中的除法有两种运算：

- 如果两个操作数都为整型变量时，结果为舍去小数取整；
- 如果两个操作数中有一个为小数（即浮点型变量）时，结果为正常的小数。

（2）输出“x%y”时，输出的格式“x%%y=%d”。因为“%”为格式化输入/输出时的起始字符，所以要在屏幕上输出“%”可以连续打两个“%”，就可以在屏幕上显示一个“%”字符。



表达式是由运算符将运算对象（如常量、变量和函数调用等）连接起来的具有合法语义的句子。在 C 语言中，由于运算符比较丰富，加之引入了赋值、复合赋值和逗号运算符，因而可以构成灵活多样的表达式。这些表达式的应用，一方面可以将程序设计得短小简洁，另一方面还可以完成某些在其他高级程序设计语言中较难实现的运算功能。

C 语言中的运算符主要分为以下几类：算术运算符、关系运算符、逻辑运算符、赋值运算符、位运算符、条件运算符、逗号运算符、求字节运算符、取地址运算符、指针运算符等。

1．算术运算符及算术表达式

【思考题 2-2】中，和、差、积、商与余数都代表了算术运算。表 2-3 列出了 C 语言提供的算术运算符及其功能。

表 2-3　C 语言的算术运算符及其功能

运　算　符	名　　称	运 算 类 型	示　　例	功　　能
+	正号运算符	单目运算符	+5	取正数 5
−	负号运算符	单目运算符	−5	取负数 5
+	加法运算符	双目运算符	a+b	求 a 与 b 之和
−	减法运算符	双目运算符	a−b	求 a 与 b 之差
*	乘法运算符	双目运算符	a*b	求 a 与 b 之积
/	除法运算符	双目运算符	a/b	求 a 与 b 之商
%	求余运算符	双目运算符	a%b	求 a 除以 b 的余数
++	自增运算符	单目运算符	++a	将 a 的值加 1
−−	自减运算符	单目运算符	−−a	将 a 的值减 1

说明：

1）求余运算符“%”

“%”又称取模运算符，要求其两侧必须为整型数，它的作用是取两个整型数相除的余数，**余数的符号与被除数的符号相同**。例如【思考题 2-2】中的 e=x%y，x 的值为 9，y 的值为 4，因此 e=x%y 相当于 e=9%4，它的结果是 1。现在假设 x 的值为−9，则−9%4 的结果就是−1，如果 x 不变，y 为−4，则−9%−4 的结果是−1。

2）除法运算符“/”

当两个操作数都是整数时，运算的结果是整数（舍去小数取整），即表示“整除”；如果参加运算的两个数中有一个是实数，则结果是实数。在【思考题 2-2】中，x 与 y 分别是 9 和 4，是两个整数，x/y 的结果是 2；但如果把 x 定义为 float　x=9.0，则 x/y 的结果就变为 2.250000。

3）自增运算符“++”或自减运算符“−−”

自增运算符“++”或自减运算符“−−”的作用是使运算对象的值增 1 或减 1。它们既可以做前缀运算符（位于运算对象的前面），例如++x 和−−x，也可以做后缀运算符（位于运算对象的后面），例如 x++和 x−−。使用自增运算符或自减运算符，应注意以下几个问题。

（1）使用自增或自减运算符只适合于整型或字符型变量，而不能用于常量或表达式，例如（x+y）++和++9 都是不合法的。

（2）自增或自减运算符，通常用于循环语句中，使循环语句的值加 1 或减 1；也可以用于指针变量，使指针变量指向下一个地址或前一个地址。

（3）在只需对变量本身进行加 1 或减 1 而不考虑表达式值的情况下，前缀运算和后缀运算的效果完全相同，否则，结果是有不一样的。

① 当自增（减）运算符单独为一个表达式时：自加（减）符号在前或后一样，都是将该变量自加（减）1。例如：

```
++i;  ←等价于→  i++;  ←等价于→  i=i+1;
--i;  ←等价于→  i--;  ←等价于→  i=i-1;
```

② 当自增（减）运算符用在表达式时：如果自加（减）符号在变量前面，则表示该变量要先自加（减），然后再参与整个表达式的运算，且运算时是整个表达式从左至右将全部自增（减）运算完成后再参与表达式的运算。如果自加（减）符号在变量后面，则表示该变量要先参与整个表达式的运算，然后再进行自加（减）。

（4）自增（减）运算符的结合性是“自右向左”。下面举例说明。例如，如果有定义语句：

```
int  i=2,j;
```

则语句 j=−i++；的运行结果是什么？

解：分析如下：

- ++、−−、+（正号）、−（取负）是同级运算符，结合方向为自右向左；
- −i++等价于−(i++)；
- 对于括号内的自增运算，要先使用 i，再使 i 增加 1。

所以语句的运算结果是：i 的值为 3，j 的值为−2。

又如，有以下定义语句：

```
int  i=3,j;
```

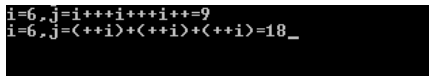
则语句：j=(++i)+(++i)+(++i)；和 j=i+++i+++i++；（两条语句运行之前 i 值都是 3）运行结果是什么？

解：分析如下：

- 第一个语句 j=(++i)+(++i)+(++i);自增符号在变量 i 的前面，所以变量 i 先自加 3 次（3+1+1+1 等于 6），再将三个 i 相加 6+6+6=18。所以 j 的值等于 18，i 值等于 6。

➢ 第二个语句 j=i+++i+++i++;等价于 j=(i++)+(i++)+(i++);（因为自增符号是自右向左结合的），所以自增符号在变量 i 的后面。所以再将三个 i 相加 3+3+3 等于 9，然后再将变量 i 自加 3 次（3+1+1+1 等于 6）。所以 j 值等于 9，i 值等于 6。

通过以上两个语句可以看出，在两个语句中，i 的结果都是 6，而 j 的结果却因为自增符号在前和在后有所不同，读者需注意理解。程序运行的结果如下：



由算术运算符或圆括号将运算对象（操作数）连接起来的有意义的式子称为算术表达式。在【思考题 2-2】中，x+y，x−y，x*y，x/y 等都是算术表达式，还有 x++，−−x 或复杂一些的如 x*(y%c)−0.9+ 'A'等都是合法的算术表达式。



2．赋值运算符与赋值表达式

“=”是 C 语言的赋值运算符，在【思考题 2-2】中，语句 a=x+y 中“=”就是赋值符号，而不是我们数学意义上的“等于号”。数学上的“等于号”（相当于关系运算符中的“比较等于”）在 C 语言中用“==”表示。

C 语言允许在赋值运算符“=”之前加上其他运算符，构成复合赋值运算符。C 语言共有 10 种复合赋值运算符，如表 2-4 所示。

表 2-4　C 语言的复合赋值运算符

运　算　符	名　　称	示　　例	等　价　于
+=	加赋值运算符	a+=b	a=a+b
−=	减赋值运算符	a−=b	a=a−b
=	乘赋值运算符	a=b	a=a*b
/=	除赋值运算符	a/=b	a=a/b
%=	取余赋值运算符	a%=b	a=a%b
&=	位与赋值运算符	a&=b	a=a&b
\|=	位或赋值运算符	a\|=b	a=a\|b
∧=	位异或赋值运算符	a∧=b	a=a∧b
<<=	左移赋值运算符	a<<=b	a=a<>=	右移赋值运算符	a>>=b	a=a>>b

在表 2-4 中，前 5 种是复合赋值算术运算符，后 5 种是复合赋值位运算符。有关位运算符的知识，将在第 9 章中介绍。



（1）赋值运算符和复合赋值运算符的结合方向均为从右到左，优先级只高于逗号运算符，而比其他运算符的优先级都低。例如：表达式 x*=y+2 等价于 x=x*(y+2)。

赋值表达式是由赋值运算符“=”将一个变量和表达式连接起来的式子，其一般格式为：

```
变量名=表达式
```

（2）赋值运算符左边必须是变量，赋值表达式的值就是被赋值后的变量值。如果一个语句中出现多个复合赋值表达式时，从右向左依次进行赋值。

例如，如果有定义语句 int a=12;，则执行完语句 a+=a−=a*=a;后，a 的值是多少？

解：按照从右至左的次序对表达式进行计算，首先计算 a*=a 相当于 a=a*a，即 a=12*12=144，然后再计算 a=a−a=144−144=0，然后再计算 a=a+a=0，所以 a 的值为 0。（注意里面的运算是每次将计算后的值先赋给变量 a，然后再将变化后的 a 参与下一次运算）。

再如，求两数之和，程序代码如下：

```
main()
{  int  x=9,y=4;
   x+=y;
   printf("x+y=%d",x);
}
```

运行结果如下：

```
x+y=13
```

二、关系运算、逻辑运算与条件运算

【思考题 2-3】 输入一个字母，判断它是否是小写字母，若是则转换成大写字母，否则不转换，并输出所得的结果。

程序分析

在输入一个字符时，先判断它的取值区间。大写字母 A～Z 的 ASCII 值是 65～90，小写字母 a～z 的 ASCII 码值是 97～122。大小写字母 ASCII 码的差值为 32，因此把小写字母转换成大写字母只需将其 ASCII 码减去 32 即可，反之将大写字母转换成小写字母只需将其 ASCII 码加上 32 即可。

编写程序代码

```
main()
{   char  c;                              /*定义变量 c 为字符数据类型*/
    printf("\ninput char:");
    scanf("%c",&c);                       /*输入字符 c*/
    c=(c>='a'&&c<='z')?c-32:c;            /*若 c 在'a'～'z'范围内,转换成大写字母*/
    printf("\noutput char:%c\n",c);       /*输出转变后的结果 c*/
}
```

调试运行程序

程序运行结果如下：

```
input char:r
output char:R
```

小讲堂

3．关系运算与关系表达式

在【思考题 2-3】中，c>='a'表示了字符 c 与字母 a 之间的大小关系，其中“>=”就是一

个关系运算符。关系运算符用于比较两个运算对象的大小。C 语言提供的关系运算符如表 2-5 所示。

表 2-5　C 语言的关系运算符

运 算 符	名　称	运 算 类 型	示　例	功　能
<	小于运算符	双目运算符	a<b	判断 a 是否小于 b
<=	小于等于运算符	双目运算符	a<=b	判断 a 是否小于等于 b
>	大于运算符	双目运算符	a>b	判断 a 是否大于 b
>=	大于等于运算符	双目运算符	a>=b	判断 a 是否大于等于 b
==	等于运算符	双目运算符	a==b	判断 a 和 b 是否相等
!=	不等于运算符	双目运算符	a!=b	判断 a 和 b 是否不相等

使用关系运算符时，应注意以下几个问题。

（1）在上述 6 个运算符中，前 4 个运算符的优先级高于后 2 个运算符的优先级。

（2）应将等于关系运算符“==”与赋值运算符“=”相区分。“==”是关系运算符，用于比较运算；而“=”是赋值运算符，用于赋值运算。

（3）关系运算符的优先级低于算术运算符而高于赋值运算符。它们的结合性是自左至右。

由关系运算符将两个对象连接起来的表达式就是关系表达式，如程序中的 c>='a'就是一个关系表达式。其运算对象可以是常量、变量或表达式。关系表达式的运算结果为逻辑值，只有两种：“真”或“假”。在 C 语言里用 1 表示“真”（非零值也认为是“真”），用 0 表示“假”。

例如，有如下定义：

```
char  c='d';
int  m=2,n=5;
```

求下列各表达式的值：

① c+1=='e'

② c+'A'−'a'!='D'

③ m−2*n<=n+9

④ m==2<n

解：代码分析如下：

① 字符数据的比较按照其 ASCII 码进行。“+”的优先级大于“==”，因此先进行 c+1 运算。c 代表'd'，则 c+1 的 ASCII 码值为 101；再进行 101=='e'的运算，由于'e'的 ASCII 值为 101，因此该等式成立，为“真”，故表达式 c+1=='e'的值为 1。

② 先计算 c+'A'−'a'，结果为 68；再进行 68!='D'的运算，'D'的 ASCII 值为 68，故该表达式不成立，所以 c+'A'−'a'!= 'D'的值为 0。

③ 先进行 m−2*n 的运算，结果为−8；然后再计算表达式 n+9，结果为 14；最后进行 m−2*n<=n+9 的运算，该表达式成立，故其值为 1。

④ 因为“<”的优先级大于“==”，所以要先进行 2<n 的运算，结果为 1；再进行 m==1 的运算，结果为 0，表达式 m==2<n 的值为 0。

小讲堂

4．逻辑运算符与逻辑表达式

在【思考题 2-3】中的代码 c>='a'&&c<='z'，表示 c>='a'并且 c<='z'，其中“&&”属于逻辑运算符。逻辑运算符用来对运算对象进行逻辑操作，C 语言提供的逻辑运算符及其功能如表 2-6 所示。

表 2-6　C 语言的逻辑运算符及其功能

运 算 符	名　　称	运 算 类 型	示　　例	功　　能
!	逻辑非	单目运算符	!a	若 a 为真，则!a 为假，否则!a 为真
&&	逻辑与	双目运算符	a&&b	若 a，b 均为真，则 a&&b 为真，否则 a&&b 为假
‖	逻辑或	双目运算符	a‖b	若 a，b 均为假，则 a‖b 为假，否则 a‖b 为真

使用逻辑运算符，应注意以下几个问题。

（1）三个逻辑运算符的优先次序为：!（逻辑非）→&&（逻辑与）→‖（逻辑或），即逻辑非“!”最高，逻辑与“&&”次之，逻辑或“‖”最低。

（2）逻辑非“!”的优先级高于算术运算符，逻辑与“&&”和逻辑或“‖”的优先级低于算术运算符和关系运算符，高于赋值运算符。因此在 c>='a'&&c<='z'中，要先进行“c>='a'”和“c<='z'”的关系运算，再进行逻辑与运算。

（3）逻辑运算符中逻辑非“！”的结合方向是自右至左，逻辑“&&”和逻辑或“‖”的结合方向是自左至右。

【思考题 2-3】中“c>='a'&&c<='z'”是一个逻辑表达式，所谓逻辑表达式是用逻辑运算符将运算对象连接起来的式子。逻辑表达式运算结果也有两种：“真”或“假”。在 C 语言中用 1 表示“真”，用 0 表示“假”。如表 2-7 所示为逻辑运算的真假值表。

表 2-7　逻辑运算的真假值表

a	b	!a	!b	a&&b	a‖b
真	真	假	假	真	真
真	假	假	真	假	真
假	真	真	假	假	真
假	假	真	真	假	假

我们再来看一看【思考题 2-3】，如果假设输入的字符 c 是“m”，m∈[a，z]区间，则 c>='a'成立，为真，而 c<='z'也成立，为真，表达式 c>='a'&&c<='z'结果为真；否则，如果输入的字符是“M”，则 c>='a'成立，为真，而 c<='z'不成立，为假，表达式 c>='a'&&c<='z'结果为假。

在逻辑表达式的求值过程中，并不是所有的运算对象都参加运算，而是按其运算对象自左至右的计算顺序，当某个运算对象的值计算出来后，可以确定整个逻辑表达式的值时，其余的运算对象将不再参加计算。例如：

① a&&b&&c：如果 a 为假，就不必判别 b 和 c 的值；如果 a 为真，b 为假，则不必判别 c 的值；只有 a 和 b 都为真时才判别 c 的值。

② a||b||c：如果 a 为真，就不必判别 b 和 c 的值；如果 a 为假，b 为真，则不必判别 c 的值，如果 a 和 b 都为假时才判别 c 的值。

③ !a||b&&c：因为逻辑非的优先级最高，因此要先计算!a，而逻辑与的优先级大于逻辑或，所以第二步要进行逻辑与 b&&c 的计算，最后再进行逻辑或的计算。



5．条件运算符与条件表达式

条件运算符是由字符“?”和“:”组成的，要求有三个运算对象，是 C 语言中唯一的三目运算符。条件运算符的优先级高于赋值运算和逗号运算符，而低于其他运算符，其结合性为自右至左。

条件表达式是由条件运算符将运算对象连接起来的式子，其一般格式为：

```
表达式 1？ 表达式 2： 表达式 3
```

条件表达式的求解过程为：先求解表达式 1，若表达式 1 的值为 1（真），则求解表达式 2，并将其作为整个表达式的值；如表达式 1 的值为 0（假），则求解表达式 3，并将其作为整个表达式的值。

在【思考题 2-3】中，如果输入字符 c 的值为“m”，则表达式 1(c>='a'&&c<='z')成立，为真，执行表达式 2（即 c−32），并把表达式 2 的结果赋值给变量 c；如果输入字符 c 的值为“M”，则表达式 1(c>='a'&&c<='z')不成立，为假，执行表达式 3，把表达式 3 的结果赋值给变量 c。

读者可以思考一下，求两个数中的最大值，如果用条件表达式表示，应该怎样写该语句？

三、圆括号运算符、逗号运算符和sizeof运算符

在 C 语言中还有圆括号运算符、逗号运算符和 sizeof 运算符。

【思考题 2-4】 设 a、b、c 三个整数变量，初始值分别为 1、2 和 3，计算 a*(b+c)的值，要求使用逗号运算符来写出表达式。



在这个数学算式中，应该先计算 b+c 的值，再将和与 a 相乘，最后输出结果。我们将赋值表达式和计算表达式都写在一起，构成逗号表达式，并将这个表达式的结果赋给变量 s。注意书写格式和最后赋值是哪个表达式。




```
main()
{   int  s,a,b,c;
    s=(a=1,b=2,c=3,a*(b+c));
    printf("a=%d\nb=%d\nc=%d\na*(b+c)=%d\n",a,b,c,s);
}
```



程序运行结果如下：



小讲堂

6．圆括号运算符

圆括号运算符“()”的优先级最高，用它将某些运算符和运算对象括起来以后，这些括起来的运算符和运算对象要优先运算。

例如【思考题 2-4】中，代码 s=(a=1,b=2,c=3,c=a*(b+c));，尽管运算符“*”的优先级比运算符的“+”的优先级高，但由于（b+c）使用了圆括号运算符，故应优先运行 b+c。在这个语句中，使用了两个“()”运算符，应先计算最里面的圆括号，再计算外面的。如果两个“()”并列排列，则应遵循由左向右的优先原则。

小讲堂

7．逗号运算符

s=(a=1,b=2,c=3,c=a*(b+c));中的“,”即逗号运算符。在 C 语言中，符号“,”除了做分隔符外，还可以作为运算符将若干个表达式连接在一起形成逗号表达式。

逗号表达式的一般格式为：

表达式 1,表达式 2,…,表达式 n

逗号表达式的运算规则是：先求解表达式 1，再求解表达式 2，依次求解到表达式 n，最后一个表达式的值就是整个逗号表达式的值。逗号运算符的优先级最低，结合性为自左至右。例如 s=(a=1,b=2,c=3,c=a*(b+c));是一个赋值语句，它是将“=”右边括号内的表达式的值赋给左边的变量 s，该语句可以改写成以下形式：

```
a=1;
b=2;
c=3;
c=a*(b+c);
s=c;
```

最后得到结果 5，s 值也为 5。

小讲堂

8．求字节运算符sizeof

1）sizeof 的定义格式

sizeof（数据类型名） 或 sizeof（变量名）

2）功能

测定某一种类型数据所占存储空间长度，结果是该类型在内存中所占的字节数。括号内可以是该数据类型名或是该类型的变量名。例如：

```
int  x;
x=sizeof(int);        /* 语句执行的结果是 x 的值为 2 */
```

也可以写为：

```
sizeof(x)              /*函数的返回值也是 2 */
```

又例如，编写一个程序，求出字符型、短整型、基本整型、长整型、单精型浮点型、双精度浮点的内存字节数。程序编写如下：

```
main()
{  int  a,b,c,d,e,f;
   a=sizeof(char);
   b=sizeof(short);
   c=sizeof(int);
   d=sizeof(long);
   e=sizeof(float);
   f=sizeof(double);
   printf("sizeof(char)=%d\n",a);
   printf("sizeof(short)=%d\n",b);
   printf("sizeof(int)=%d\n",c);
   printf("sizeof(long)=%d\n",d);
   printf("sizeof(float)=%d\n",e);
   printf("sizeof(double)=%d\n",f);
}
```

程序运行结果如下：

```
sizeof(char)=1
sizeof(short)=2
sizeof(int)=2
sizeof(long)=4
sizeof(float)=4
sizeof(double)=8
```

小讲堂

9．数据类型的转换

在 C 语言中，数据类型的转换方式有两种：自动类型转换和强制类型转换。

所谓自动类型转换是指在 C 语言中，不同的数据可以出现在同一个表达式中。在进行运算时，C 语言自动进行必要的数据类型转换，以完成表达式的求值。当与一个运算符相关联的两个运算对象的类型不同时，其中的一个运算对象的类型将转换成与另一个运算对象的类型相同，转换的优先顺序如图 2-3 所示。

例如：表达式 4+5L−3.5 求值过程中的类型转换可示意如下：

```
4+5L-3.5→4L+5L-3.5→9L-3.5→9.0-3.5→5.5
```

其中，4 首先转换成 long 型后完成加法运算，其中 long 型的结果再转换成 double 型后完成减法运算，最后的结果是 double 型。

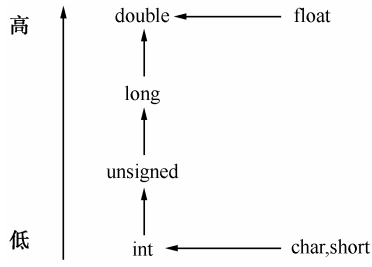


图 2-3　自动类型转换的规则

注意

（1）上面的转换规则不适用于位运算，因为位运算的运算对象只能是整型数据，若实际处理的是实型数据，编译系统不会实现自动类型转换，而是报告出错。

（2）上面的转换规则也不适用于赋值运算或复合赋值运算，对于这两种运算，以运算符左边变量的类型为标准进行转换。例如，执行下列程序段：

```
int x=5;
```

```
float y=3.85;
x+=y;
printf("x=%d\n",x);
```

结果输出 8，说明变量 y 的小数部分在转换成 int 型时被舍去了。

自动类型转换是系统自动进行的，不需要用户干预。C 语言还允许用户根据自己的需要将运算对象的数据类型转换成所需要的数据类型，这就是强制类型转换。强制类型转换的运算格式如下：

（类型标识符）运算对象　　或　　类型标识符（运算对象）

例如：

```
(int)3.14              /*将 3.14 转换成整型，其值为 3*/
(int)(3.14+4.78)       /*将表达式 3.14+4.78 的和 7.92 转换成 int 型，值为 7*/
(int)3.14+4.78         /*将 3.14 转换成 int 型的值 3，然后再加上 4.78，值为 7.78*/
```

练一练

【练习 2-5】 假设 a、b、c 均为整数，用 C 语言描述下列命题。

（1）a 小于 b 或小于 c。

（2）a 或 b 都大于 c。

（3）a 和 b 中至少有一个小于 c。

（4）a 是非正数。

（5）a 是奇数。

（6）a 不能被 b 整除。

解：描述如下：

（1）a<b||a<c

（2）a>c||b>c

（3）(a<c&&b<c)||(a<c&&b>c)||(a>c&&b<c)

（4）a<=0

（5）a%2!=0 或 a%2==1

（6）a%b!=0

【练习 2-6】 假设 x=3，y=z=4，求下列表达式的值。

（1）(z>=y&&z>=x)?1∶0

（2）z>=y&&y>=x

解：描述如下：

（1）1

（2）1

本讲小结

本讲主要讲述了 C 语言的运算符，不同的运算符具有不同的优先级和结合性。表达式是由运算符连接常量、变量、函数所组成的式子。每个表达式都有一个值，表达式求值按运算符的优先级和结合性所规定的顺序进行。数据类型是可以转换的，转换的方式有两种：自动类型转换和强制类型转换。

想一想

在本讲中，我们介绍了大量的运算符，并且知道每个运算符都有优先级，自己试着总结一下所有的运算符的优先级。

第四讲　数据的输入与输出

学习目标

- 掌握字符输入函数 getchar 的使用方法；
- 掌握字符输出函数 putchar 的使用方法；
- 掌握标准输入函数 scanf 的使用方法；
- 掌握标准输出函数 printf 的使用方法。

一、字符输入、输出函数

【思考题 2-5】 从键盘上输入一个字符，将该字符存到计算机内并显示在屏幕上，并输出一个换行符。然后再输入一个字符并直接输出（不存入任何变量），再输出一个回车换行符。

程序分析

从键盘输入一个字符并显示到屏幕，需要用到单个字符的输入函数和字符输出函数。注意这两个函数的使用格式和参数形式。

编写程序代码

```
main()
{  char  ch;
   printf("Please input two characters: ");
   ch=getchar();                /*从键盘输入一个字符并赋给变量 ch */
   putchar(ch);
putchar('\n');
   putchar(getchar());          /*从键盘输入一个字符并直接输出*/
   putchar('\n');
  }
```

调试运行程序

从键盘上输入两个字符 a 和 b，程序运行结果如下：

```
Please input two characters: ab
a
b
```

小讲堂

1．字符输入函数getchar

1）字符输入函数 getchar()的格式

getchar()　或　getch()

2）功能

函数 getchar()的功能是从键盘上接收输入的一个字符。

3）说明

调用函数时，当程序执行到 getchar 函数调用语句时，将等待输入，只有当用户输入字符，并按回车键后，才接收输入的第 1 个字符，并在屏幕上回显该字符，同时送到内存的缓冲区，准备赋给指定的变量。并且对空格符、制表符（Tab 键）和回车换行符（Enter 键）都被当做有效字符读入。getch()函数是立即接收用户来自键盘上的输入，不把字符回显到屏幕上。例如：

```
char  c;
 ⋮
c=getchar();
```

其中 c 是字符型或整型变量。



2．字符输出函数putchar

1）字符输出函数 putchar()的定义格式

```
putchar(ch)
```

2）功能

其作用是将参数代表的字符通过标准输出设备（通常是显示器）输出。

3）说明

参数 ch 可以是字符常量、字符变量或整型表达式，其功能等价于：

```
printf("%c",ch);          /*标准输出函数 printf()会在本节后面讲解*/
```

putchar 函数也可以输出一些特殊字符（控制字符），如在【思考题 2-5】中的语句：putchar('\n');，作用是输出一个“回车换行”字符。

putchar()函数的参数也可以是一个函数，如【思考题 2-5】中语句：putchar(getchar());，就是用 getchar()从键盘中读入一个字符，然后直接使用该字符作为 putchar()函数的参数，输出到屏幕上。

二、标准输入、输出函数

【思考题 2-6】 分析下列程序，观察运行结果。



源程序如下：

```
main()
{  char  ch='a';                                          /*定义变量并赋值 */
int  a=98;
unsigned  b=1000;
long  c=123456789;
float  x=3.14;
double  y=1.2345678;
printf("(1)a=%d,a=%c,ch=%d,ch=%c\n",a,a,ch,ch);      /*A*/
printf("(2)b=%u\n",b);                                 /*B*/
printf("(3)c=%ld\n",c);                                /*C*/
```

```
    printf("(4)x=%f,y=%f\n",x,y);                    /*D*/
    printf("(5)x=%e,y=%e\n",x,y);                    /*E*/
    printf("(6)y=%-10.2f\n",y);                      /*F*/
    }
```

程序分析

变量 a 为整型，所以语句 A 前两项是输出变量 a 的整数值和对应其 ASCII 码的字符；变量 ch 为字符型，语句 A 后两项输出变量 ch 的整数值和对应其 ASCII 码的字符。因为字符在计算机中是以 ASCII 码形式存储的，所以可以以整数和字符两种形式输出。

变量 b 为无符号整型变量，1000 为正整数，所以语句 B 直接原样输出 b。

变量 c 为长整型变量，123456789 的大小不超过长整型变量的最大值，所以语句 C 原样输出 c。

变量 x 为单精度浮点型变量，变量 y 为双单精度浮点型变量，所以在输出时默认是小数点后保留 6 位，所以语句 D 以%f 格式输出 x 和 y 时保留小数点后 6 位，初始化值不足 6 位由系统自动补足 6 位。

语句 E 是将变量 x 和 y 以科学计数法形式输出，e（或 E）前的小数为整数位只有一位的纯小数，e 或 E 后为科学计数法的 10 的次幂数。

语句 F 是将变量 y 以 10 位数，保留小数点后两位数的形式输出。

调试运行程序

程序运行结果如下：

```
(1)a=98,a=b,ch=97,ch=a
(2)b=1000
(3)c=123456789
(4)x=3.140000,y=1.234568
(5)x=3.14000e+00,y=1.23457e+00
(6)y=1.23
```

小讲堂

3．格式输入函数scanf

1）格式控制函数 scanf 的调用格式

```
scanf("格式控制字符串",输入地址列表);
```

2）功能及说明

scanf 函数用来输入任何类型数据，可同时输入多个类型或不同类型的数据。括号内的两个参数功能如下。

（1）“格式控制字符串”说明。格式字符串的一般形式为：

```
%[*][输入数据宽度][长度]类型
```

其中有方括号[]的项为任选项（可写可不写）。各项的意义如下。

① 类型：表示输入数据的类型，其格式符和意义如表 2-8 所示。

表 2-8　scanf 函数格式控制字符

格　式	字 符 意 义
d	输入十进制整数

续表

格　式	字 符 意 义
o	输入八进制整数
x	输入十六进制整数
u	输入无符号十进制整数
f 或 e	输入实数（用小数形式或指数形式）
c	输入单个字符
s	输入字符串

例如，程序段：

```
char  ch;
scanf("%c",&ch);
```

程序运行时输入为：A↙时，就已经将字符'A'存到变量 ch 中。

② “*”符：用以表示该输入项，输入后不赋予相应的变量，即跳过该输入值。

例如有如下输入语句：

```
scanf("%d%*d%d",&a,&b);
```

当输入为：1　2　3 时，将 1 赋予 a，2 被跳过，3 赋予 b。

又如，有一小程序如下：

```
main()
{  int  a;
   char  c;
   printf("input  data:");
   scanf("%3d%*1d%3c",&a,&c);
   printf("%d,%c",a,c);
}
```

运行程序，输入

```
12345,abc↙
```

程序输出结果是：

```
123,5
```

这是因为在格式控制字符中，第一个“%3d”说明读入一个 3 位的整数，则将 123 读入并赋给变量 a；第二个“%*1d”说明读入一个 1 位的整数，将 4 读入，但*表示读入该数据却不存入任何变量，所以这个整数 4 没被赋值；第三个“%3c”说明要读入字符（按 3 位读取），但因为字符只占一位，所以所给格式错误，只按字符默认格式读入 1 位，将字符 5 读入，并存入变量 c，所以程序运行结果是 123,5。

③ 输入数据宽度：用十进制整数指定输入的宽度（即字符数）。

例如：

```
scanf("%5d",&a);
```

输入：

```
12345678↙
```

只把 12345 赋予变量 a，其余部分被截去。

又如：

```
scanf("%4d%4d",&a,&b);
```

输入：

```
12345678↙
```

将把 1234 赋予 a，而把 5678 赋予 b。

④ 长度：长度格式符为 l 和 h，l 表示输入长整型数据（如%ld）和双精度浮点数（如%lf），h 表示输入短整型数据。

（2）地址列表中给出各变量的地址，地址是由地址运算符“&”后跟变量名组成的。

在 C 语言中，使用了地址这个概念，这是与其他语言不同的。应该把变量的值和变量的地址这两个不同的概念区别开来，变量的地址是 C 编译系统分配的，用户不必关心具体的地址是多少。

例如，&a 和&b 分别表示变量 a 和变量 b 的地址，这个地址是编译系统在内存中为变量 a、b 分配的地址。

在赋值语句中给变量赋值，如：

```
a=567;
```

则 a 为变量名，567 是变量的值，&a 表示变量 a 的地址。但在赋值号左边是变量名，不能写地址。而 scanf 函数在本质上也是给变量赋值，但要求写变量的地址，如&a。这两者在形式上是不同的。&是一个取地址运算符，&a 是一个表达式，其功能是求变量的地址。

3）使用 scanf 函数须注意以下几点

（1）在用 scanf 函数输入数据时，不能规定精度。例如，在输入时不能设浮点型数据的小数位数，如：

```
float  f;
scanf("%7.2f",&f);
```

是错误的，只能写成：

```
scanf("%f",&f);
```

（2）scanf 中要求给出变量地址，若给出变量名则会出错。例如 scanf("%d",a);是非法的，应改为 scnaf("%d",&a);。

（3）在输入多个数值数据时，若格式控制串中没有非格式字符做输入数据之间的间隔，则可用空格、Tab 或回车。C 编译在碰到空格、Tab、回车或非法数据（如对“%d”输入“12A”时，A 即为非法数据）时即认为该数据结束。

（4）在输入字符数据时，若格式控制串中无非格式字符，则认为所有输入的字符均为有效字符。例如：

```
scanf("%c%c%c",&a,&b,&c);
```

输入为：（“□”表示空格）

```
d□e□f↙
```

则把'd'赋予 a，'□'（一个空格字符）赋予 b，'e'赋予 c。只有当输入为：

```
def↙
```

时，才能把'd'赋予 a，'e'赋予 b，'f'赋予 c。

如果在格式控制中加入空格作为间隔，例如：

```
scanf ("%c %c %c",&a,&b,&c);
```

则输入时各数据之间可加空格。

（5）如果格式控制串中有非格式字符，则输入时也要输入该非格式字符。例如：

```
scanf("%d,%d,%d",&a,&b,&c);
```

其中用非格式符“,”做间隔符，故输入时应为：

```
5,6,7↙
```

又如：

```
scanf("a=%d,b=%d,c=%d",&a,&b,&c);
```

则输入应为：

```
a=5,b=6,c=7↙
```

（6）注意用 scanf 函数（使用“%s”格式）读入字符串时，如果输入的字符串有空格或 Tab 键时，只将空格之前的所有字符读入，之后的字符串并不读入。所以用这种格式不能读入带空格的字符串。如果想读入带空格的字符串可使用 gets 函数（在第 4 章的字符数组有讲解）。

（7）在 scanf 函数中不使用%u 说明符，对 unsigned 型数据，以%d、%o 或%x 格式输入。



4．格式输出函数printf

1）printf 函数调用格式

```
printf("格式控制字符串",输出项列表);
```

或

```
printf("字符串");
```

printf 中的输出项是和前面的格式控制字符串按顺序一一对应的。“项”可以是常量、变量、表达式、数组和函数调用，其值应和格式说明相容。

2）格式控制字符串。格式字符串的一般形式

```
%[标志][输出最小宽度][.精度][长度]类型
```

其中有方括号[]的项为任选项。各项的意义如下。

（1）类型：类型字符用以表示输出数据的类型，其格式符和意义如表 2-9 所示。

表 2-9　printf 函数的控制字符列表

格 式 字 符	意　　义
d	按十进制形式输出带符号的整数（正数前无+号）
o	按八进制形式无符号输出（无前导 0）
x	按十六进制形式无符号输出（无前导 0x）
u	按十进制无符号形式输出
c	按字符形式输出一个字符
f	按十进制形式输出单、双精度浮点数（默认 6 位小数）
e	按指数形式输出单、双精度浮点数
s	输出以\0 结尾的字符串
ld	长整型输出
lo	长八进制整型输出
lx	长十六进制整型输出
lu	按无符号长整型输出
m 格式字符	按宽度 m 输出，右对齐
-m 格式字符	按宽度 m 输出，左对齐
m,n 格式字符	按宽度 m，n 位小数，或截取字符串前 n 个字符输出，右对齐
-m,n 格式字符	按宽度 m，n 位小数，或截取字符串前 n 个字符输出，左对齐

注意 表 2-8 中的所有格式字符必须小写。

printf 函数中的“输出项列表”部分由表达式组成，这些表达式应与“格式控制字符串”中的格式说明符的类型一一对应，若“输出项列表”中有多个表达式，则每个表达式之间应由逗号隔开，各输出项可以是任意合法的表达式（包括常量、变量和函数调用）。因此 printf 函数也具有计算的功能。

```
printf ("%d\n", 100) ;                    /*输出显示 100*/
printf ("%d\n", 12340000+5678) ;          /*输出显示：12345678*/
printf("%f\n",(x=123.0)+(y=0.4567)) ;     /*输出显示：123.456700*/
printf("%6.2f\n", 123.4567) ;             /*输出显示：123.46*/
printf("%.3f\n",123.4567) ;               /*输出显示：123.457*/
```

（2）标志：标志字符为-、+、空格、0 和#五种，其意义表 2-10 所示。

表 2-10 printf 函数的控制字符的附加说明

标 志 符	意 义
-	结果左对齐，右边填空格
+	输出符号（正号或负号）
空格	输出值为正时冠以空格，为负时冠以负号
0	用 0（零）去填充域宽（多用在右对齐方式中在数据前补 0）
#	对 c，s，d，u 类无影响；对 o 类，在输出时加前缀 0；对 x 类，在输出时加前缀 0x；对 e，g，f 类当结果有小数时才给出小数点

（3）输出最小宽度：用十进制整数来表示输出的最少位数。若实际位数多于定义的宽度，则按实际位数输出，若实际位数少于定义的宽度则补以空格或 0。

（4）精度：精度格式符以“.”开头，后跟十进制整数。本项的意义是：如果输出数字，则表示小数的位数；如果输出的是字符，则表示输出字符的个数；若实际位数大于所定义的精度数，则截去超过的部分。

（5）长度：长度格式符为 h、l 两种，h 表示按短整型量输出，l 表示按长整型量输出。

3）输出项列表

printf 函数中的“输出项列表”部分由表达式组成，这些表达式应与“格式控制”字符串中的格式说明符的类型一一对应，若“输出项列表”中有多个表达式，则每个表达式之间应由逗号隔开。

4）说明

（1）printf 函数中的“格式控制”字符串中的每一个格式说明符，都必须与“输出项列表”中的某一个变量相对应。如，整型变量应与“%d”对应，“%f”与单精度浮点型变量对应。

若要显示“%”字符，则应在“格式控制”字符串中连写两个“%”，如：

```
printf("x=%d%%",100/4);
```

将显示：x=25%。

（2）对格式说明符 c、d、s 和 f 等，可以指定输出字段的宽度。

① md：m 为指定的输出字段的宽度。如果数据的位数大于 m，则按实际位数输出，否则输出时向右对齐，左端补以“空格”符。

② mc：m 为指定的输出字段的宽度。若 m 大于一个字符的宽度，则输出时向右对齐，

左端补以“空格”符。

③ ms：m 为输出时字符串所占的列数。如果字符串的长度（字符个数）大于 m，则按字符串的本身长度输出；否则，输出时字符串向右对齐，左端补以“空格”符。

④ -ms：m 的意义同上。如果字符串的长度小于 m，则输出时字符串向左对齐，右端补以“空格”符。

⑤ m.nf：m 为浮点数据所占的总列数（包括小数点），n 为小数点后面的位数。如果数据的长度小于 m，则输出时向右对齐，左端补以“空格”符。

⑥ -m.nf：m、n 的意义同上。如果数据的长度小于 m，则输出向左对齐，右端补以“空格”符。

（3）除了格式说明符及其输出字段的宽度外，在“格式控制”字符中的其他字符，将按原样输出。

（4）在显示数据时，可以不指定输出字段的宽度，而直接利用系统隐含的输出宽度。

注意 浮点型数据在赋值的过程中，float 型的变量 x 只能接收 7 位有效数字，而 double 型的变量 y 能接收 15～16 位有效数字。如果赋值的小数字后的位数过多或过少，都会造成数据的精度丢失或不足。

我们再来分析下面这个程序的输出结果，注意其中的浮点型数据。

```
main()
{  float  x;
   double  y;
   x=11111.1111;
   y=11111.1111;
   printf("x=%f\ny=%f\n",x,y);
}
```

程序运行结果如下：

```
x=11111.111328
y=11111.111100
```

x 和 y 的初值只有 4 位小数，输出结果却有 6 位小数，这是因为“%f”作为格式字符控制实数输出时，系统自动将整数部分全部输出，小数部分输出 6 位。

（5）printf 函数的输出项中如果有自增（减）运算符，则从右向左依次输出每次自增（减）的值。请仔细分析以下程序的运行结果。例如：

```
main()
{  int  i=5;
   printf("\n%d,%d,%d",++i,++i,++i);
   i=5;
   printf("\n%d,%d,%d",i++,i++,i++);
}
```

程序运行结果是：

```
8,7,6
7,6,5
```

分析如下：

当 i=5 时，第一行输出函数：

```
printf("\n%d,%d,%d",++i,++i,++i);
```

输出项有三个，且是一个变量 i 的三个自增语句（自增符号在变量前表示先计算自增再输出其值）。输出函数 printf 是从最右的一个++i 进行计算得到值为 6，输出 6；然后再向左计算中间的++i，进行计算得到值为 7，输出 7；然后再对最左边的++i 进行计算得到值为 8，输出 8。所以第一行的输出结果为 8，7，6。

当 i=5 时，第二行输出函数：

```
printf("\n%d,%d,%d",i++,i++,i++);
```

输出项也有三个，且是一个变量 i 的三个自增语句（自增符号在变量后表示先输出其值再计算自增）。输出函数 printf 是从最右的一个 i++开始，先输出 i 值 5，然后 i 自加变为 6；对于中间的 i++，先输出 i 值 6，再计算自增进行计算得到值为 7；再处理最左边的 i++，先输出 i 值 7，然后计算 i++得到 i 值为 8（但并不输出其值）。所以第二行的输出结果为 7，6，5。

练一练

【练习 2-7】 分析下列程序的输出结果，注意其中的数据类型转换。

```
main()
{  int  b;
   long  c;
   float  f=123.456;
   b=12;
   c=1234567;
   printf("%d,%4d,%-4d,%04d\n",b,b,b,b);
   printf("%ld,%pld,%-pld,%0pld\n",c,c,c,c);
   printf("%s,%10s,%-10s,%10.4s,%-10.4s\n","string","string","string",
   "string","string");
   printf("%10.2f,%-10.2f,%.2f\n",f,f,f);
}
```

解：在输出函数中，%md 之间的数字 m 表示按 m 位来输出该数据；%0md 表示按 m 位来输出该数据，不足的位补 0；%–mld 表示按 m 位来输出长整型数据，“–”号表示按左对齐方式输出（默认是右对齐方式输出）。程序运行结果如下：

```
12,  12,12  ,0012
1234567,  1234567,1234567  ,001234567
string,    string,string    ,      stri,stri
   123.46,123.46    ,123.46
```

【练习 2-8】 分析下列程序的输出结果，注意其中的数据类型转换。

```
main()
{   int  i;
    long  int  l;
    float  f;
    double  d;
    i=l=f=d=100/3;
    printf("i=%-10dl=%-10ldf=%-15fd=%-15f\n",i,l,f,d);
    i=l=f=d=(double)100/3;
    printf("i=%-10dl=%-10ldf=%-15fd=%-15f\n",i,l,f,d);
    i=l=f=d=(double)100000/3;
    printf("i=%-10dl=%-10ldf=%-15fd=%-15f\n",i,l,f,d);
    d=f=l=i=(double)100000/3;
    printf("i=%-10dl=%-10ldf=%-15fd=%-15f\n",i,l,f,d);
}
```

解：将 double 型数据赋给 float 型变量时，将截取 7 位有效数字；将 float 型数据赋给长整型变量时，舍去小数部分，只将整数部分赋给长整型变量；将长整型数据赋给整型变量时，截取低 16 位赋给整型变量，且最高位将被看做符号位。

将有符号整型数据赋给长整型变量时，将按符号位扩展，保持数值大小不变；将整型数据赋给 float 型变量时，数值大小保持不变，但先转换为 float 型数据再赋给左边的变量；将 float 型数据赋给 double 型变量时，数值大小保持不变，但有效数字扩展为 16 位。

程序运行结果如下：

```
i=33        l=33        f=33.000000      d=33.000000
i=33        l=33        f=33.333332      d=33.333333
i=-32203    l=33333     f=33333.332031   d=33333.333333
i=-32203    l=-32203    f=-32203.000000  d=-32203.000000
```

【练习 2-9】 输入一个三位的整数，分解它的符号、百位数字、十位数字和个位数字，然后依次输出。

解：首先通过条件语句判断该整数是否大于 0，如果大于 0，则将“+”号赋给 c4；如果小于 0，则将“-”号赋给 c4。然后对该数取绝对值，将该值除以 10 取余数，得到该数的个位。将该数除以 10 再赋给 x，得到该数去掉个位的结果（如原来 x 是 123，得到 12）。同样方法，再将新得到的 x 同样先除 10 取余数得到十位上的数字。最后将 x 除 10 取商，得到该数的百位数字。最后输出正、负符号及每位上的数字。

源程序如下：

```
#include "stdio.h"
#include  <math.h>                          /*程序中使用了数学函数 abs*/
main()
{  char  s;                                 /*定义存放符号位的字符变量*/
   int  x,a,b,c;
   printf("please input a number:");        /*输入提示信息*/
   scanf("%d",&x);                          /*键盘输入 x 的值*/
   s=x>=0?'+':'-';                          /*将 x 的符号赋 c*/
   x=abs(x);                                /*取 x 的绝对值*/
   c=x%10;                                  /*求得 x 的个位数字*/
   x=x/10;                                  /*去掉个位数,取出 x 的高位*/
   b=x%10;                                  /*求得 x 的十位数字*/
   a=x/10;                                  /*求得 x 的百位数字*/
   printf("%c\n%d\n%d\n%d\n",s,a,b,c); /*输出符号位,个位,十位,百位*/
}
```

程序运行结果如下：

```
please input a numer:-125
The result:
-
1
2
5
```

本讲小结

本讲主要介绍了字符输入/输出函数，格式化输入/输出函数。一个完整的 C 程序应该可以实现数据的输入和输出。注意各个函数的书写格式和其中参数的含义。

想一想

自增（减）运算符的结合性是怎样的？自增（减）运算符放在变量前和后有什么区别？

在输入整型数据时，采用什么格式输入？输入字符串采用什么格式？输入浮点型数据采用什么格式？如果想输出一行提示语，应该采用什么语句？

本章小结

本章主要讲述了 C 语言的标识符、变量及常量、基本数据类型及类型间的相互转换、常用的运算符及其优先级和结合性、表达式、数据的输入/输出函数等。

所谓标识符，是指用来标识程序中用到的变量名、函数名、类型名、数组名、文件名，以及符号常量名等的有效字符序列。C 语言中的标识符包括三类：用户自定义标识符、关键字和预定义标识符。

常量和变量是 C 语言中的两种重要的数据组织形式，常量又称常数，是指在程序运行过程中其值不能被改变的量。C 语言中的常量分为字面常量和符号常量。变量是指程序运行过程中其值可以被改变的量。

C 语言中的数据类型包括基本类型、构造类型和空类型，其中基本类型数据包括整型数据、实型数据和字符数据。

C 语言提供了丰富的运算符，不同的运算符具有不同的优先级和结合性。

表达式是由运算符连接常量、变量、函数所组成的式子。每个表达式都有一个值，表达式求值按运算符的优先级和结合性所规定的顺序进行。

数据的类型是可以转换的，转换的方法有两种：自动类型转换和强制类型转换。

本章还介绍了 C 程序的字符输入函数 getchar、字符输出函数 putchar、格式输入函数 scanf 和格式输出函数 printf。

课后习题二

一、选择题

1．C 语言中，运算对象必须是整型数的运算符是_____。

A）%　　B）/　　C）%和/　　D）*

2．C 语言中最简单的数据类型包括_____。

A）整型、实型、逻辑型　　B）整型、实型、字符型

C）整型、字符型、逻辑型　　D）整型、实型、逻辑型、字符型

3．一个 C 语言的语句至少应包一个_____。

A）{}　　B）逗号　　C）分号　　D）什么都不要

4．设 x 和 y 均为 int 型变量，则语句：x+=y;y=x−y;x−=y;的功能是_____。

A）把 x 和 y 按从大到小排列　　B）把 x 和 y 按从小到大排列

C）无确定结果　　D）交换 x 和 y 中的值

5．若 a 为 int 类型，且其值为 3，则执行完表达式 a+=a−=a*a 后，a 的值是_____。

A）−3　　B）9　　C）−12　　D）6

6．若有以下程序段：

```
int c1=1,c2=2,c3;
c3=1.0/c2*c1;
```

则执行后，c3 中的值是_____。

A）0　　B）0.5　　C）1　　D）2

7. 设 a、b、c、d、m、n 均为 int 型变量，且 a=5、b=6、c=7、d=8、m=2、n=2，则逻辑表达式(m=a>b)&&(n=c>d)运算后，n 的值为_____。

A）0　　B）1　　C）2　　D）3

8．设 x=3，y=−4，z=6，表达式!(x>y)+(y!=z)||(x+y)&&(y−z)的结果为_____。

A）0　　B）1　　C）−1　　D）6

9．设有如下的变量定义：

```
int  i=8,k,a,b;
unsigned  long  w=5;
double  x=1.42,y=5.2;
```

则以下符合 C 语言语法的表达式是_____。

A）a+=a−=(b=4)*(a=3)　　B）x%(−3)　　C）a=a*3=2　　D）y=float(i)

10．运算完下面的 C 语言程序段以后，a、b、c 的值分别是_____。

```
int  x=10,y=9;
int  a,b,c;
a=(--x==y++)?--x:++y;
b=x++; c=y;
```

A）6　9　13　　B）8　7　11　　C）8　8　10　　D）8　7　10

11．在 C 语言中，合法的字符常量是_____。

A）'\084'　　B）'\x43'　　C）'ab'　　D）"\0"

12．字符（char）型数据在计算机内存中的存储形式是_____。

A）反码　　B）补码　　C）EBCDIC 码　　D）ASCII 码

13．x、y、z 被定义为 int 型变量，若从键盘给 x、y、z 输入数据，正确的输入语句是_____。

A）input x,y,z;　　B）scanf("%d%d%d",&x,&y,&z);

C）scanf("%d%d%d",x,y,z);　　D）read("%d%d%d",&x,&y,&z);

14．请读程序：

```
int  i=0,j=0,a=6;
if((++i>0)||(++j>0))  a++;
printf("i=%d,j=%d,a=%d\n",i,j,a);
```

则上面程序的输出结果是_____。

A）i=0,j=0,a=6　　B）i=1,j=0,a=7　　C）i=1,j=1,a=6　　D）i=1,j=1,a=7

15．请读程序：

```
int  a,b,c;
a=(b=(c=10)+5)-5;
printf("a,b,c=%d,%d,%d",a,b,c);
c=a=0; b=(a+10);
printf("a,b,c=%d,%d,%d",a,b,c);
```

则上面程序的输出结果是_____。

A）a,b,c=0,10,10 a,b,c=10,15,10　　B）a,b,c=10,15,10 a,b,c=10,15,10

C）a,b,c=10,15,10 a,b,c=0,10,0　　D）a,b,c=10,15,10 a,b,c=10,15,15

二、填空题

1．若有定义：char　c='\010';，则字符变量 c 中包含的字符个数为_____。

2．以下程序段的输出结果是_____。

```
main()
```

```
{  int  x,y,z;
   x=1;y=2;z=3;
   x=y==z<=x||x+y!=z;
   printf("%d,%d",x,y);
}
```

3．有以下定义和语句：

```
char  c1= 'b',c2= 'e';
printf("%d,%c",c2-c1,c2-'a'+'A');
```

则输出结果是_____。

4．以下程序段的输出结果是_____。

```
main()
{  int  a,b,c;
   a=b=c=1;
   ++a&&++b||++c;
   printf("%d,%d,%d",a,b,c);
}
```

5．如果 a=1,b=2,c=3,d=4，则条件表达式 a>b?a:c>d?c:d 的值是_____。

6．以下程序的输出结果是_____。

```
main()
{  int  a=5;
   printf("%d",!a);
}
```

7．以下程序的输出结果是_____。

```
main()
{  int i=5,j=5,x,y;
   x=i++;
   y=++j;
   printf("i=%d,x=%d\n",i,x);
   printf("j=%d,y=%d\n",j,y);
}
```

三、编程题

1．输入一个整数，求该数的绝对值。

2．计算下列表达式的值，假设各表达式之间没有影响。已知：int x=7,y=4;。

（1）!x&&y−−

（2）x||y&&x+y

（3）++x,++y,x+y

（4）x=3,y=8,x<y?x++:y++

（5）x+=y%=x+y

3．设计一个程序，已知球的半径 r=2，求球的体积 v（$v=4/3\times\pi\times r^3$，π取 3.14）。

4．用条件表达式编写一段程序，设 a=1，b=2，c=3，输出 a、b、c 三个数中最大的数。

第三章　C语言的流程控制

上一章介绍的表达式是计算机语言处理数据的基本手段。但是对于一个程序来讲，它是由一个个语句组成的。从语言的语法角度看，计算机程序设计语言与自然语言类似，常量、变量、运算符及表达式是组成语句的“单词”，而语句则是组成程序的“单位”。从程序设计角度来看，用户是通过语句来描述程序的解题过程，即通过语句来控制程序的执行流程。任何程序均由以下三种基本结构组成：

（1）顺序结构：从流程上是从前向后顺序执行；

（2）选择结构：根据判断条件的结果有选择地执行；

（3）循环结构：有条件地重复执行某一过程。

其中，选择结构的语句有 if 语句和 switch 语句，循环结构的语句有 while、do-while 和 for 三种基本循环语句，还有 break 和 continue 语句和流程转向 goto 语句也多用到循环结构中。

本章将详细介绍如何在 C 程序中实现选择结构。同时，在程序设计中对于那些需要重复执行的操作应该采用循环结构完成，利用循环结构处理各类重复操作既简单又方便。在 C 语言中有三种可以构成循环的循环语句，还包括流程转向语句和循环语句的嵌套。通过本章的学习，应该掌握以下内容：

- if 语句的三种定义格式及使用；
- switch 语句的定义格式及使用；
- while、do-while 和 for 循环语句的结构及应用；
- 循环语句嵌套结构及流程控制语句的使用；
- break、continue 及 goto 语句的使用。

第五讲　选择结构



学习目标

- 掌握 if 语句的三种定义格式;
- 掌握 if 语句的嵌套;
- 掌握 Switch 语句的格式及使用实例。

一、基本if语句

【思考题 3-1】 从键盘上输入一个整数，如果大于等于零输出“Positive　Number”（正数），小于零输出“Negative　Number”（负数）。要求用基本 if 语句实现。

程序分析

（1）从键盘输入一个整数；

（2）判断该数是大于等于零或者小于零；

（3）输出对应结果。

其流程图如图 3-1 所示。解决该程序的方法利用了基本 if 语句结构，在“小讲堂”中将对基本 if 语句结构做详细说明。

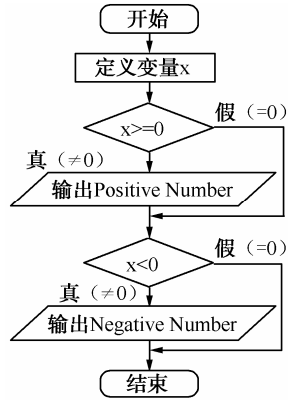


图 3-1 【思考题 3-1】的程序流程图

编写程序代码

```
main()
{  int  x;                  /*定义整型变量 x*/
   printf("input x:");  /*提示从键盘输入一个数*/
   scanf("%d",&x);      /*接受输入的数赋值给 x*/
   if(x>=0)             /*判断 x 是否大于 0*/
     printf("%d is Positive Number!\n",x);   /*如果大于 0，输出结果为正数*/
   if(x<0)                                   /*判断 x 是否小于 0*/
     printf("%d is Negative Number!\n",x);   /*如果小于 0，输出结果为负数*/
}
```

调试运行结果

输入 x 的值为 5，输出“5 is Positive Number!”；再次运行程序，输入一个 x 的值为-3，输出“-3 is Negative Number!”。

程序运行结果如下：

```
input x:5
5 is Positive Number!
input x:-3
-3 is Negative Number!
```

小讲堂

1．if语句基本形式

1）基本 if 语句定义格式

```
if(表达式)
    语句;
```

例如：

```
if(a<b)
{ t=a; a=b; b=t; }
```

其中，if 是 C 语言的关键字，表达式两侧的圆括号不可少，一条“语句”称为 if 子句。如果在 if 子句中需要多条语句，则应该使用花括号把一组语句括起来组成复合语句，这样在语法上仍满足“一条语句”的要求。

2）if 语句的执行过程

首先计算紧跟在 if 后面一对圆括号中表达式的值，如果表达式的值为非零（“真”），则执行其后的 if 子句，然后去执行 if 语句后的下一个语句。如果表达式的值为零（“假”），则跳过 if 子句，直接执行 if 语句后的下一个语句。

基本 if 语句的流程示意图如图 3-2 所示。

3）说明

（1）if 语句自动结合一个语句，当满足条件需要执行多个语句时，应用一对大括号{}将需要执行的多个语句括起，形成一个复合语句。

（2）if 语句中表达式形式很灵活，可以是常量、变量、任何类型表达式、函数、指针等。只要表达式的值为非零值，条件就为真，反之条件为假。

二、标准if语句

【思考题 3-2】 从键盘上输入一个数，如果大于等于零输出“Positive Number”（正数），否则输出“Negative Number”（负数）。要求用标准 if 语句实现。

程序分析

（1）从键盘输入一个数；

（2）判断该数是否大于等于零；

（3）输出结果。

其流程图如图 3-3 所示。解决该程序的方法利用了标准 if 语句结构，在“小讲堂”中将对标准 if 语句结构做详细说明。

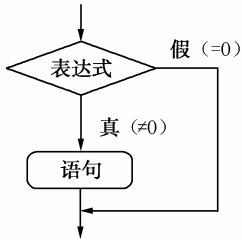


图 3-2 基本 if 语句的流程示意图

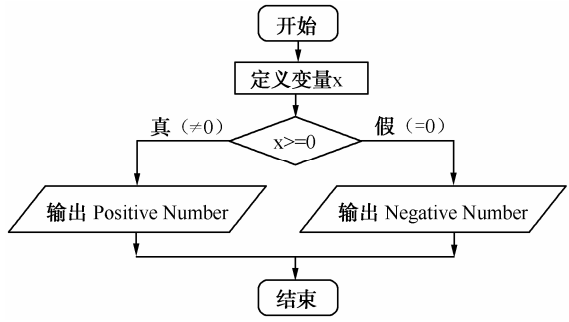


图 3-3 【思考题 3-2】的程序流程图

编写程序代码

```
main()
{  int  x;
   printf ("input x:");
   scanf("%d",&x);
   if(x>=0)                                    /*判断 x 是否大于 0*/
     printf("%d is Positive Number!\n",x);     /*如果 x 大于 0，输出结果为正数*/
   else                                        /*如果 x 大于 0 的条件不成立*/
     printf("%d is Negative Number!\n",x);     /*输出 x 小于 0 的结果为负数*/
}
```

调试运行结果

输入 x 的值为 5，输出“5 is Positive Number!”；再次运行程序，输入一个 x 的值为−3，输出“−3 is Negative Number!”。

程序运行结果如下：

```
input x:5
5 is Positive Number!
input x:-3
-3 is Negative Number!
```

小讲堂

2．if语句标准形式

1）标准 if 语句定义格式

```
if(表达式)
语句 1;
else
语句 2;
```

例如：

```
if(a!=0)
printf("a!=0\n");
else
printf("a==0\n");
```

在这里，“语句 1”称为 if 子句，“语句 2”称为 else 子句，这些子句只允许是一条语句，若需要多条语句时，则应该使用花括号把这些语句括起来组成复合语句。注意，else 不是一条独立的语句，它只是 if 语句的一部分，不允许有这样的语句：

```
else  printf("***");
```

在程序中 else 必须与 if 配对，共同组成一条 if-else 语句。

2）if-else 语句的执行过程

首先计算紧跟在 if 后面圆括号内表达式的值。如果表达式的值为非零，执行 if 子句，然后跳过 else 子句，去执行 if 语句后的下一条语句；如果表达式的值为零，跳过 if 子句，去执行 else 子句，接着去执行 if 语句后的下一条语句。

3）说明

（1）if 后面圆括号中的表达式，可以是任意合法的 C 语言表达式（如逻辑表达式、关系表达式、算术表达式、赋值表达式等），也可以是任意类型的数据（如整型、实型、字符型等）。

（2）无论是否有 else 子句，if 子句中如果只有一条语句，则此语句后的分号不能省略。

例如：

```
if(x!=0)
   printf("%f",x);   /*此处的分号不能省略*/
else
   printf("%f",y);
```

标准 if 语句的流程示意图如图 3-4 所示。

三、复合if语句

【思考题 3-3】 从键盘上输入一个整数，如果大于零输出“Positive Number”（正数），如果小于零输出“Negative Number”（负数），否则输出“Zero”（零）。

程序分析

（1）从键盘输入一个整数；

（2）判断该数是大于零、小于零还是等于零；

（3）输出结果。

流程图如图 3-5 所示。

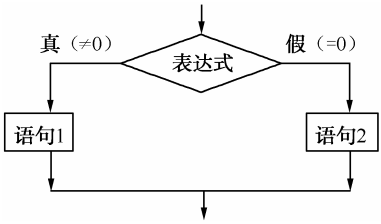


图 3-4　标准 if 语句的流程示意图

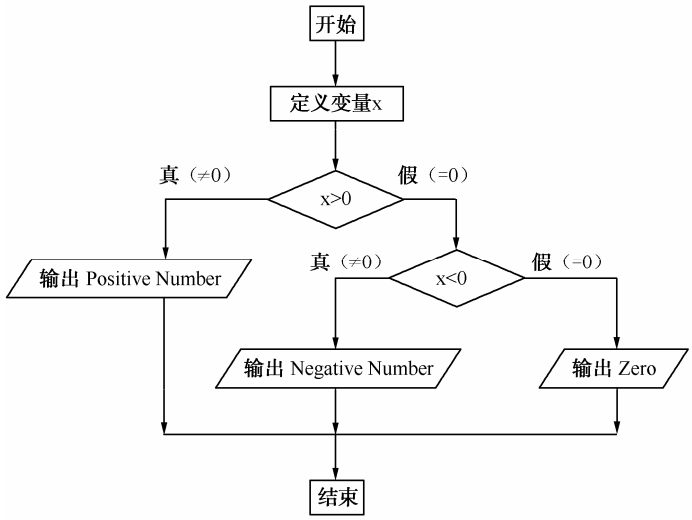


图 3-5　【思考题 3-3】的程序流程图

编写程序代码

```
main()
{  int  x;
   printf ("input x:");
   scanf("%d",&x);
   if(x>0)                                  /*判断 x 是否大于 0*/
     printf("%d is Positive  Number!\n",x);  /*如果 x 大于 0，输出结果为正数*/
   else  if(x<0)                            /*否则继续判断 x 是否小于 0*/
     printf("%d is Negative  Number!\n",x);  /*如果 x 小于 0，输出结果为负数*/
   else
     printf("\n%d is Zero!\n",x);           /*否则输出结果为零*/
}
```

调试运行结果

输入 x 的值为 5，输出“5　is　Positive　Number!”；再次运行程序，输入一个 x 的值为-3，输出“-3　is　Negative　Number!”；第三次运行程序，输入 x 的值为 0，输出“0 is Zero！”。程序运行结果如下：

```
input x:5
5 is Positive  Number!
input x:-3
-3 is Negative  Number!
input x:0
0 is Zero!
```

小讲堂

3．if语句复合形式

1）复合 if 语句定义格式

```
if(表达式 1)
    语句 1;
else   if(表达式 2)
    语句 2;
    …
else   if(表达式 n)
    语句 n;
else
    语句 n+1;
```

2）复合 if 语句的执行过程

首先计算表达式 1 的值，若表达式的值为“真”（表达式具有非 0 值），则执行语句 1；否则继续判断表达式 2，若表达式的值为“真”，则执行语句 2；否则继续判断下一个条件，……直到最后前面 n 个条件都不成立时，执行 else 后面的语句 n+1。

复合 if 语句的流程图如图 3-6 所示。

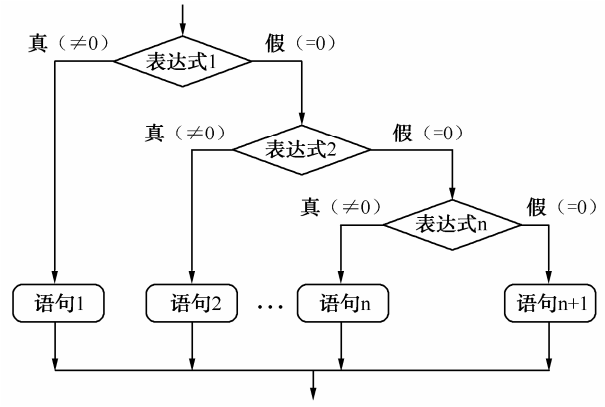


图 3-6　复合 if 语句的流程图

四、if语句的嵌套

小讲堂

4．if语句的嵌套

1）嵌套的概念

在 if 语句中，语句 1 和语句 2 本身也可以是 if 语句，此时称为 if 语句的嵌套。

2）嵌套的格式

if 语句可以内嵌在 if 子句中，又可以内嵌在 else 子句中。

例如：

```
if ( )
   if ( )
     语句 1;          ┐
   else               ├ 内嵌 if
     语句 2;          ┘
else
```

```
if ( )
  语句3;
else
  语句4;
```
内嵌 if

3）嵌套的说明

以上形式的嵌套 if 语句执行过程可以这样理解：从上向下逐行对 if 后的表达式进行检测。当某一个表达式的值为非零时，就执行与此有关子句中的语句，阶梯形中的其余部分被越过去。如果所有表达式的值都为零，则执行最后的 else 子句；此时，如果程序中最内层的 if 语句没有 else 子句，即没有最后的那个 else 子句，那么将不进行任何操作。



注意 C 语言的语法规定，else 子句总是与前面最近的未曾配对（不带 else）的 if 组成一对。

五、switch语句

用标准 if 语句只能进行两路选择，但在实现多路选择时须使用复合的 if 语句，因此用 if 语句解决多路问题有时不太方便，这时可利用 switch 语句实现多条件多分支程序设计。

【思考题 3-4】 设计一个程序，用 switch 语句实现由键盘输入一个成绩，输出该成绩的等级。其中 A 代表 90 分以上，B 代表 80～89，C 代表 70～79，D 代表 60～69，E 代表 60 分以下。如果成绩不在 0～100 之间则输出错误信息。



程序分析

（1）定义 score 变量，存放由键盘输入的学生成绩；

（2）输入学生成绩赋给变量 score；

（3）根据学生 score 整除 10 得到其十位上的数字，确定该成绩范围，并输出其等级。



编写程序代码

```
main()
{float  score;
printf("\nPlease input a score:");
scanf("%f",&score);
if(score>=0&&score<=100)
{ switch ((int)score/10)
  { case 10:
    case  9:printf("\n%.2f class is A!",score);break;
    case  8:printf("\n%.2f class is B!",score);break;
    case  7:printf("\n%.2f class is C!",score);break;
    case  6:printf("\n%.2f class is D!",score);break;
    default:printf("\n%.2f class is E!",score);
  }
}
else
   printf("\nThe score is wrong!");
}
```



调试运行结果

程序运行结果分析：运行 6 次程序，每次输入一个区间的成绩（其中最后一次输入一个

错误成绩，不在 0～100 之间）。

程序运行结果如下：

```
Please input a score:97

97.00 class is A!
Please input a score:80

80.00 class is B!
Please input a score:73

73.00 class is C!
Please input a score:61

61.00 class is D!
Please input a score:28

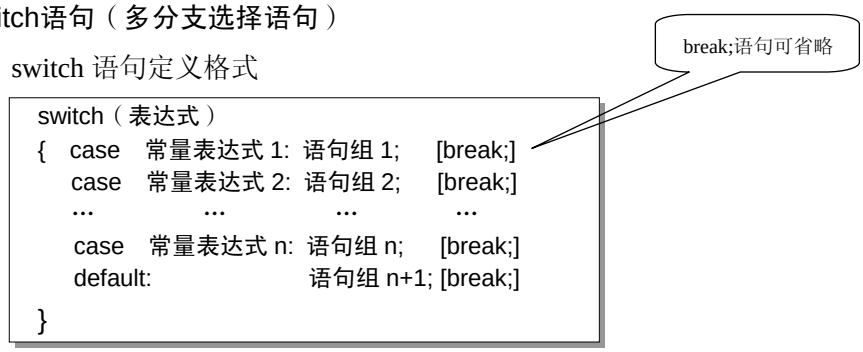
28.00 class is E!
Please input a score:130

The score is wrong!_
```



5．switch语句（多分支选择语句）

1）switch 语句定义格式



2）说明

switch 语句又称为开关语句，语句的执行过程如下：程序执行至 switch 语句首先对括号内的表达式进行计算，然后按顺序找出与常量表达式值相匹配的 case，以此作为入口，执行 case 语句后面的各个语句组，直到遇到 break 或 switch 语句的右花括号终止语句。如果没有任何一个 case 能与表达式值相匹配，则执行 default 语句后的语句组，若 default 及其后语句组省略，则不执行 switch 中任何语句组，而继续执行下面的程序。

3）使用 switch 语句注意的问题

（1）default 语句及后面语句组可以省略。

（2）switch 后圆括号内表达式可以是整数表达式、字符表达式或枚举表达式。case 后面是整数或字符，也可以是不含变量和函数的常量表达式。同一个 switch 语句中的 case 后的值不能相同。

（3）case 及后语句和 default 及后语句出现次序可以任意。

（4）case 与其后的常量表达式之间要有空格，否则系统会识别不出该常量表达式。

（5）执行完一个 case 语句后，程序自动转到后面的语句执行，直到遇到 break 或 switch 语句的右花括号终止语句。

练一练

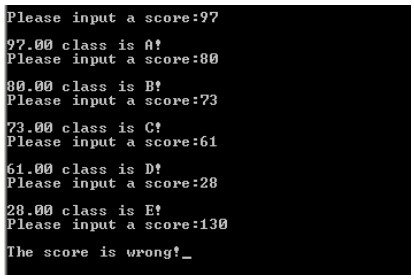
【练习 3-1】 从键盘上输入一个成绩（0～100 之间），输出这个分数对应的等级。其中 A 代表 90 分以上，B 代表 80～89，C 代表 70～79，D 代表 60～69，E 代表 60 分以下，如果成绩不在 0～100 之间则输出错误信息。要求用复合 if 语句来实现。

解：源程序如下：

```
main()
{  float  score;
   printf("\nPlease input a score:");
   scanf("%f",&score);
   if(score>=0&&score<=100)
   {  if(score>=90)
        printf("\n%.2f class is A!",score);
      else if(score>=80)
        printf("\n%.2f class is B!",score);
      else if(score>=70)
        printf("\n%.2f class is C!",score);
      else if(score>=60)
        printf("\n%.2f class is D!",score);
      else
        printf("\n%.2f class is E!",score);
   }
   else
      printf("\nThe score is wrong!");
}
```

程序运行结果分析：运行 6 次程序，每次输入一个区间的成绩（其中最后一次输入一个错误成绩，不在 0～100 之间）。

程序运行结果如下：

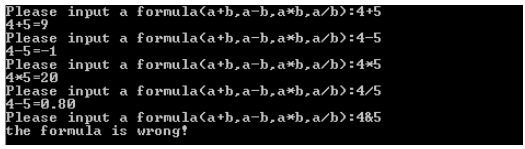


【练习 3-2】 从键盘上输入一个数学四则运算表达式（a+b、a−b、a*b 或 a/b），要求计算出该表达式的值。

解：分析题意，应该设两个整型变量 a 和 b，再设一个存放运算符的字符型变量 ch，然后根据 ch 的值来进行相应的运算，将结果输出即可。程序如下：

```
main()
{  int  a,b;
   char  ch;
   printf("\nPlease input a formula(a+b,a-b,a*b,a/b):");
   scanf("%d%c%d",&a,&ch,&b);
   switch(ch)
   {  case  '+':printf("%d+%d=%d",a,b,a+b);break;
      case  '-':printf("%d-%d=%d",a,b,a-b);break;
      case  '*':printf("%d*%d=%d",a,b,a*b);break;
      case  '/':printf("%d/%d=%.2f",a,b,(float)a/b);break;
      /*因为结果为小数，所以要将 a 强制转换成小数形式，结果才正确*/
      default:  printf("the formula is wrong!");
   }
}
```

程序运行 5 次，分别输入 4 个表达式：4+5、4−5、4*5、4/5 和 4&5（错误的表达式），程序运行结果如下：



【练习 3-3】 编写一程序，从键盘上输入一个年份 year（4 位十进制数），判断其是否为闰年。闰年的条件是：能被 4 整除、但不能被 100 整除，或者能被 400 整除。

解：闰年是年份能被 4 整除但不能被 100 整除的数，或是能被 400 整除的数。我们设年份为 year，这样就构成了一个判断闰年的条件表达式如下：

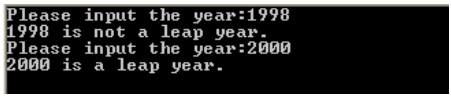
```
(year%4==0 && year%100!=0)||(year%400==0)
```

源程序如下：

```
main()
{  int  year;
   printf("Please input the year:");
   scanf("%d",&year);
   if((year%4==0 && year%100!=0)||(year%400==0))      /*判断年份是否是闰年*/
     printf("%d is a leap year.\n",year);
   else
     printf("%d is not a leap year.\n",year);
}
```

第一次输入年份“1998”，输出结果“1998 is not a leap year.”,第二次输入年份“2000”，输出结果“2000 is a leap year.”。

程序运行结果如下：



本讲小结

C 语言提供了可以进行逻辑判断的选择语句，由选择语句构成的选择结构将根据逻辑判断的结果决定程序的不同流程。选择结构是结构化程序设计的三种基本结构之一。通过本节学习，掌握 if 语句的三种格式，理解 if 语句的嵌套，掌握 switch 语句基本格式及解决问题的方法。

第六讲　循环结构

学习目标

- 掌握 while 语句的基本格式和实例;
- 掌握 do-while 语句的基本格式和实例;
- 掌握 for 语句的基本格式和实例。

在程序设计中对于那些需要重复执行的操作应该采用循环结构完成。利用循环结构处理各类重复操作既简单又方便，循环结构又称重复结构。在 C 语言中有三种可以构成循环的循环语句。

循环是一种对同一程序段有规律的重复，被重复执行的部分叫循环体。循环的执行要满足一定的条件（循环条件），循环的终止要达到一定的条件（终止条件）。在程序设计中要注

意循环不能永远运行，必须能退出循环。循环结构的特点是：循环体执行与否及其执行次数必须视其类型与条件而定，且必须能在适当的时机退出循环。

C 语言提供了 while、do-while、for 三种语句实现循环，其中 while 循环是当型循环，先判断循环条件，再根据条件决定是否执行循环体，执行循环体的最少次数为 0。

一、while语句

【思考题 3-5】 设计一个程序，用 while 循环语句实现 1～100 自然数的和。

程序分析

这是一个求 100 个数的累加和问题。加数从 1 变化到 100，可以看到加数是有规律变化的，后一个加数比前一个加数增 1，第一个加数为 1，最后一个加数为 100；因此可以在循环体中使用一个整型变量 i，每循环一次使 i 增 1，一直循环到 i 的值超过 100，用这个办法就解决了所需的加数问题。但是要特别注意的是变量 i 需要有一个正确的初值，这里初值应当设为 1。

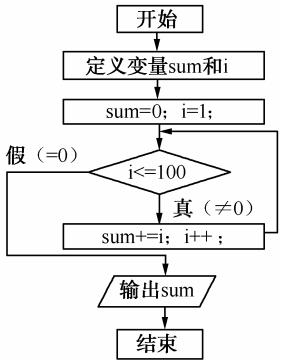


图 3-7 【思考题 3-5】的程序流程图

下一个要解决的是求累加和。设用一个变量 sum 存放这 100 个数的和值，sum 的初值为 0，可以先求 0 加 1（i 的初值为 1）的和并将其放在 sum 中，然后把 sum 中的数加上 2 再存放在 sum 中，依次类推，这和人们的心算过程没有什么区别，sum 代表着人们脑中累加的那个和数，不同的是心算过程由人们自己控制。在这里，sum 累加的过程要放在循环中，由计算机判断所加的数是否已经超过 100，加数则放在变量 i 中，并在循环过程中每一次增 1。

其流程图如图 3-7 所示。解决该程序的方法利用了 while 循环语句结构，在“小讲堂”中将对 while 循环语句结构做详细说明。

编写程序代码

```
main()
{  int  sum=0,i=1;              /*sum 的初值为 0*/
   while(i<=100)                /*当 i 小于或等于 100 时执行循环体*/
   { sum+=i; i++; }             /*在循环体中累加一次，并将 i 值自增 1*/
   printf("1+2+…+100=%d\n",sum);
}
```

调试运行结果

程序运行结果如下：

```
1+2+...+100=5050
```

小讲堂

1．While循环语句

1）while 语句基本形式

```
while(表达式)
循环体
```

while 是 C 语言的关键字。while 后圆括号中的表达式，可以是 C 语言中任意合法的表达式，由它来控制循环体是否执行。在语法上，要求循环体可以是一条简单的可执行语句；若循环体内需要多个语句，应该用大括号括起来，组成复合语句。

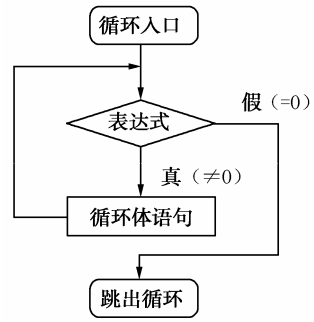


图 3-8　while 语句的程序流程图

while 语句流程图如图 3-8 所示。

2）while 语句的执行过程

计算 while 后圆括号中表达式的值。当值为非零时，执行循环体语句。执行完后再次判断表达式的值，当表达式的值为非零时，继续执行循环体；当值为零时，退出循环。

3）使用 while 语句注意的问题

（1）循环体如果包含一个以上的语句，应该用花括号括起来，以复合语句的形式出现。如果不用花括号，则 while 语句的范围只到 while 后面第一个分号处。

（2）在循环体中应该有使循环趋向于结束的语句。如无此语句，循环将永不结束。

二、do-while语句

【思考题 3-6】　设计一个程序，用 do-while 循环语句实现 1～100 自然数的和。

程序分析

（1）定义变量 sum 和 i，分别存放累计和及循环次数；

（2）累计和变量 sum 赋初值 0，循环次数 i 赋初值 1；

（3）do-while 循环求和；

（4）输出累计和结果 sum。

解决该程序的方法利用了 do-while 循环语句结构，在“小讲堂”中将对 do-while 循环语句结构做详细说明。

编写程序代码

```
main()
{  int  sum=0,i=1;
   do
   {  sum+=i;
      i++;
   }while(i<=100);
   printf("1+2+…+100=%d\n",sum);
}
```

调试运行结果

程序运行结果同【思考题 3-5】。

小讲堂

2．do-While循环语句

1）do-While 语句的基本形式

```
do
{   循环体
}while(表达式);
```

2）do-while 语句的执行过程

（1）执行 do 后面循环体中的语句。

（2）计算 while 后圆括号中表达式的值。当值为非零时，转去执行步骤①；当值为零时，结束 do-while 循环。

由 do-while 构成的循环与 while 循环十分相似，它们之间的重要区别是：while 循环控制出现在循环体之前，只有当 while 后面表达式的值为非零时，才可能执行循环体；在 do-while 构成的循环中，总是先执行一次循环体，然后再求表达式的值，因此，无论表达式的值是零还是非零，循环体至少执行一次。

注意 和 while 循环一样，在 do-while 循环体中，一定要有能使 while 后表达式的值变为 0 的操作，否则，循环将会无限制地进行下去，进入死循环。

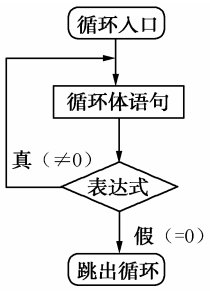


图 3-9 do-while 语句的程序流程图

3）do-while 语句的使用说明

（1）do 是 C 语言的关键字，必须和 while 联合使用。

（2）do-while 循环由 do 开始，至 while 结束。

注意 while(表达式)后的“;”不可丢，它表示 do-while 语句的结束。

（3）while 后圆括号中的表达式可以是 C 语言中任意合法的表达式，由它控制循环是否执行。

（4）按语法，在 do 和 while 之间的循环体只能是一条可执行语句；若循环体内需要多个语句，应该用大括号括起来，组成复合语句。

do-while 语句的流程图如图 3-9 所示。

三、for循环语句

for 循环语句结构简洁，使用方便，由 for 语句构成的循环按指定的次数执行循环体，它在循环体中使用一个循环变量，每重复一次后，循环变量的值会自动增加或减少。

【思考题 3-7】 设计一个程序，用 for 循环语句实现 1～100 自然数的和。

程序分析

（1）定义变量 sum 和 i，分别存放累计和及循环次数；

（2）累计和变量 sum 赋初值 0；

（3）for 循环求和；

（4）输出累计和结果 sum。

流程图如图 3-10 所示。

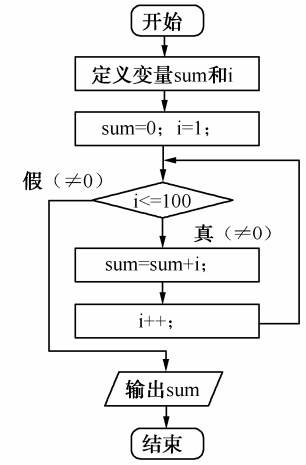


图 3-10　【思考题 3-7】的程序流程图

编写程序代码

```
main()
{  int  sum,i;
   for(i=1,sum=0;i<=100;i++)
     sum=sum+i;
    printf("1+2+3+…+100=%d\n",sum);
                                    }
```

调试运行结果

程序运行结果如下：

```
1+2+3+···+100=5050
```

小讲堂

3．for循环语句

1）for 语句基本形式

```
for(赋初值语句;条件表达式;自增(减)语句)
循环体
```

例如：

```
for(k=0; k<10; k++)
printf("*");
```

以上 for 循环在一行上输出 10 个“*”号。

for 是 C 语言的关键字，其后的圆括号中通常含有三个表达式，各表达式之间用“;”隔开。这三个表达式可以是任意形式的表达式，通常主要用于 for 循环的控制。紧跟在 for 之后的循环体，在语法上要求是一条语句；若在循环体内需要多条语句，应该用大括号括起来组成复合语句。

2）for 语句的执行过程

（1）执行“赋初值语句”为循环体变量赋初值（注意，该语句在整个循环中只在开始时执行一次）。

（2）判断“条件”是否成立：若其值为非零，转步骤③；若其值为零，转步骤⑤。

（3）执行一次 for 循环体。

（4）执行“自增（减）语句”；转向步骤②。

（5）结束循环，执行 for 循环之后的语句。

3）for 语句的使用说明

（1）for 语句中的表达式可以部分或全部省略，但两个“;”不可省略，例如：

```
for( ; ; )
printf("*");
```

三个表达式均省略，但因缺少条件判断，循环将会无限制地执行，而形成无限循环（通常称为死循环）。

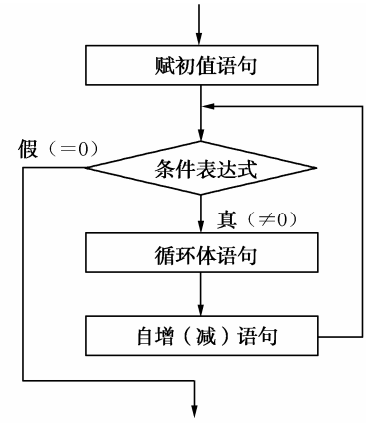


图 3-11　for 语句的流程图

（2）for 后括号中的表达式可以是任意有效的 C 语言表达式。例如：

```
for(sum=0,i=1;i<=100;sum=sum+i,i++)
{ … }
```

其中表达式 1 和表达式 3 都是一个逗号表达式。

注意　C 语言中的 for 语句书写灵活，功能较强。在 for 后的圆括号中，允许出现各种形式的与循环控制无关的表达式，虽然这在语法上是合法的，但这样会降低程序的可读性。建议初学者在编写程序时，在 for 后面的圆括号内仅含有能对循环进行控制的表达式，其他操作尽量放在循环体内完成。

for 语句的流程图如图 3-11 所示。

练一练

【练习 3-4】　分别用 while 和 do-while 两种循环语句编程实现 s=1*2*…*10 的积的程序。

解：设一个循环体变量 i 和存放乘积的变量 sum，初始设 i 等于 1，sum 等于 1。使用循环语句，将每次的 i 值乘到 sum 上，然后将 i 值自加 1，直到 i 大于 10 时循环结束（注意乘积 sum 的数值如果大于 int 型数据的取值范围时，应将其类型设为长整型 long）。

（1）使用 while 循环解题程序如下：

```
main()
{  int  i=1;
   long  sum=1;
   while(i<=10)
   {  sum*=i;  i++; }
   printf("1*2*…*10=%ld\n",sum);
}
```

程序运行结果如下：

```
1*2*...*10=3628800
_
```

（2）使用 do-while 循环解题程序如下：

```
main()
{  int  i=1;
   long  sum=1;
   do
   {  sum*=i;
      i++;
   }while(i<=10);
   printf("1*2*…*10=%ld\n",sum);
}
```

程序运行结果同使用 while 语句的程序运行结果。

注意　如果将 sum 定义为 int 型变量时，会出现溢出现象，结果错误。还要注意长整型变量 sum 在输出时格式控制符要写成“%ld”才能正确输出结果。

【练习 3-5】　编程实现从键盘输入任意数给变量 x，求从 1 到 x 的和。

解：设一个用于存放输入值的变量 x，再设一个循环体变量 i 和存放和的变量 sum。先从键盘中输入 x 的值，然后设 i 初始值等于 1，sum 等于 0。再使用循环语句，将每次的 i 值加到 sum 上，然后将 i 值自加 1，直到 i 大于 x 时循环结束。解题程序如下：

```
main()
{  int  i=1,x,sum=0;
   printf("\ninput a positive integer number:");
   scanf("%d",&x);
   while(i<=x)
   {  sum=sum+i;
      i++;
   }
   printf("1+2+…+%d=%d\n",x,sum);
}
```

输入 j 值为 5，程序运行结果如下：

```
input a positive integer number:8
1+2+...+8=36
```

本讲小结

本讲介绍了用 while、do-while、for 循环语句设计程序，注意 while 语句是先判断后执行循环，而 do-while 是执行一次循环后再判断条件，for 语句是循环结构中功能最强、最灵活、使用最多的语句。注意三种循环语句的书写格式及在什么情况下使用哪种语句。

想一想

什么情况下使用 while 语句好？什么情况下使用 do-while 语句好？它们之间的区别主要是什么？提示：在循环次数已知和未知时采用的语句不同。

第七讲　循环语句的嵌套和流程转向语句



学习目标

- 掌握循环嵌套的方法;
- 掌握流程转向语句 goto、break 和 continue 语句。

一、循环语句的嵌套

在一个循环体内又完整地包含了另一个循环，称为循环嵌套，C 语言程序可以有多层嵌套。

【思考题 3-8】　打印杨辉三角形。

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

… …

1 10 45 … … 45 10 1

程序分析

杨辉三角形中，每个数值可以由组合 C_m^n 来计算

C_0^0

C_1^0 C_1^1

C_2^0 C_2^1 C_2^2

C_3^0 C_3^2 C_3^2 C_3^3

C_4^0 C_4^1 C_4^2 C_4^3 C_4^4

… …

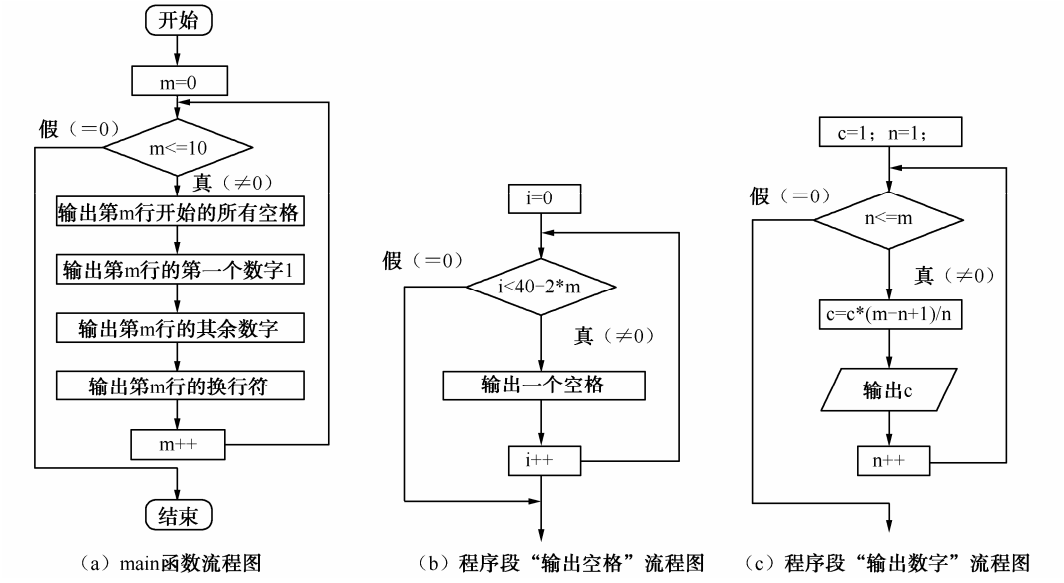
C_{10}^0 C_{10}^1 C_{10}^2 … … C_{10}^8 C_{10}^9 C_{10}^{10}

已知：

$$\begin{cases} C_m^0 = 1 & m=0,1,2,L\ 10 \\ C_m^n = (m-n+1)/n * C_m^{n-1} & n=1,2,3,L\ m \end{cases}$$

假定第一行中的数字距离左边首列为 40 个空格符，每一行的第一个值应向左移 2 个空格符，每个数字之间留 4 个空格符。

流程图如图 3-12（a）、（b）、（c）所示。



（a）main函数流程图　（b）程序段“输出空格”流程图　（c）程序段“输出数字”流程图

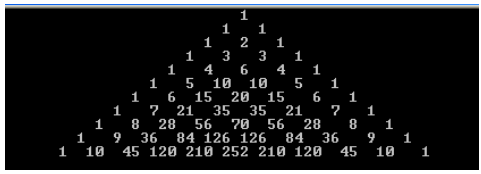
图 3-12　【思考题 3-8】的程序流程图

编写程序代码

```
main()
{  int  c,m,n,i;
   for(m=0;m<=10;m++)                          /*输出杨辉三角形*/
   {  c=1;                                     /*每行的第一个数字赋值为 1*/
      for(i=0;i<40-2*m;i++)  printf(" ");      /*输出该行左边的空格*/
      printf("%d",c);                          /*输出该行的第一个数字 1*/
      for(n=1;n<=m;n++)                        /*输出该行的其余数字*/
      {  c=c*(m-n+1)/n;
         printf("%4d",c);
      }
      printf("\n");                            /*每行结束输出换行符*/
   }
}
```

调试运行结果

程序运行结果如下：



小讲堂

1．循环语句的嵌套

一个循环体内又包含另一个完整的循环结构，称为循环的嵌套。内嵌的循环中还可以嵌套循环，这就是多层循环。

三种循环（while 循环、do-while 循环和 for 循环）可以互相嵌套。例如，以下几种都是合法的形式：

（1）while()
{ …
while()
{ … }
…
}

（2）while()
{ …
do
{ …
}while();
…
}

（3）for(;;)
{ …
for(;;)
{ … }
…
}

（4）for(;;)
{ …
while()
{ … }
…
}

（5）do
```
{  …
do
{ …
}while( );
…
}while( );
```

（6）do
```
{
  …
  for(;;)
  { … }
  …
}while( );
```

二、流程转向语句goto语句

【思考题 3-9】 设计一个程序，用 goto 语句实现 1～100 自然数的和。



程序分析

（1）定义变量 sum 和 i，分别存放累计和及循环次数；

（2）累计和变量 sum 赋初值 0，循环次数 i 赋初值 1；

（3）使用 if 语句和 goto 语句构成循环求和；

（4）输出累计和结果 sum。

解决该程序的方法利用了流程转向语句 goto 语句，在“小讲堂”中将对 goto 语句做详细说明。



编写程序代码

```
main()
{  int  sum=0,i=1;
a1:sum=sum+i;
    i++;
    if(i<=100)
       goto  a1;          /*流程转向语句，一般与 if 语句一起构成循环*/
    printf("1+2+…+100=%d\n",sum);
}
```



调试运行结果

程序运行结果如下：





2．goto语句

1）goto 语句形式

```
goto    语句标号;
```

例如：

```
goto   lable;
   …
lable:  …
```

“语句标号”用标识符表示，它的定名规则与变量名相同。

例如，在【思考题 3-9】中，a1 就是标号。一般来说，如果要使用 goto 语句构成循环，一般是用 if 语句和 goto 语句配合使用的。当 if 语句条件为真时无条件转到标号所在程序行进行执行。

2）语句执行流程

在程序执行过程中，如果遇到 goto 语句，则程序执行流程无条件地转向语句标号后的语句继续执行。

3）说明

（1）语句标号仅仅对 goto 语句有效，对其他语句不影响。

（2）同一个程序中，不允许有同名标号。

（3）goto 语句通常与条件语句配合使用，可用来实现条件转移、构成循环、跳出循环体等功能。

三、break语句

为使循环控制更加灵活，C 语言提供了 break 语句和 continue 语句，break 语句是完全从本层循环中跳出，continue 语句只是提前结束本次循环。

【思考题 3-10】 设计一个程序，判断一个整数是否为素数。

程序分析

素数就是除了 1 和它本身，不能被任何数整除的数。因此，如果一个数 x 是素数，就不能被 2～x−1 之间的任何一个数整除。进一步思考，如果 x 在 2～$\sqrt{x}$ 之间有一个数 a 可以被其整除，则在 $\sqrt{x}$ ～x−1 之间也会有一个数 b 存在，满足 x=a×b 成立。所以在查找时，只要 x 在 2～$\sqrt{x}$ 之间没有任意一个数能被整除，则 x 在 $\sqrt{x}$ ～x−1 区间内也不会有任意一个数能被整除（即循环终值为 $\sqrt{x}$ 而不是 x−1）。所以程序中使用 x%i 的余数是否为 0 来判断 x 是否能被 i 整除。退出循环后，如果 i>$\sqrt{x}$，表示 x 不能被 2～$\sqrt{x}$ 中任何一个数整除，则 x 是素数，否则 x 不是素数。

流程图如图 3-13 所示。

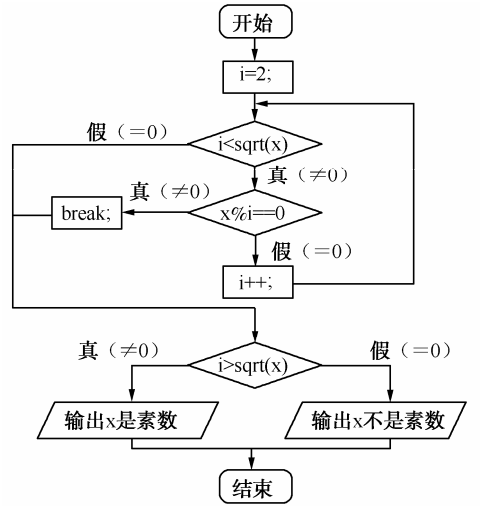


图 3-13 【思考题 3-10】的程序流程图

编写程序代码

```
#include "math.h"
main()
{int i,x;
printf("\nPlease input a integer number(>1):");
scanf("%d",&x);
for(i=2;i<=sqrt(x);i++)
  if(x%i==0)
    break;
if(i>sqrt(x))
  printf("\n%d is a prime number!",x);
else
  printf("\n%d is not a prime number!",x);
}
```

调试运行程序

程序运行结果如下：

```
Please input a integer number(>1):61
61 is a prime number!
```

小讲堂

3．break循环语句

1）break 语句基本形式

```
break;
```

2）break 语句的使用说明

（1）只能在循环体内和 switch 语句体使用 break 语句。

（2）当 break 出现在循环体中的 switch 语句体内时，其作用只是跳出该 switch 语句。当 break 出现在循环体中，但并不在 switch 语句内时，则在执行 break 后，跳出本层循环体。

3）使用 break 语句注意的问题

（1）循环体中 break 语句只能退出所在循环，不能退出整个程序。

（2）break 语句只能用于 switch 和循环语句，不能用于其他。

四、continue语句

【思考题 3-11】　设计一个程序，求 1～100 中所有奇数和。

程序分析

（1）定义存放结果变量 sum 和循环次数变量 i。

（2）给 sum 赋初值 0。

（3）循环中使用 if 判断是否为奇数，若为奇数则参与求和运算。

（4）输出 sum 的值。

流程图如图 3-14 所示。

编写程序代码

```
main()
{  int  sum,i;
   sum=0;
   for(i=1;i<=100;i++)
   {  if(i%2==0)
        continue;
      sum+=i;
   }
   printf("\n1+3+5+…+99=%d\n",sum);
}
```

调试运行程序

程序运行结果如下：

```
1+3+5+···+99=2500
_
```

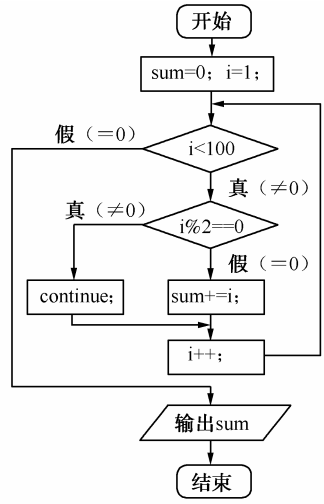


图 3-14 【思考题 3-11】的程序流程图

小讲堂

4．continue循环语句

1）continue 基本形式

```
continue;
```

2）continue 语句的使用说明

其作用是结束本次循环，即跳过本次循环体中余下尚未执行的语句，接着再一次进行循环的条件判定。在 while 和 do-while 循环中，continue 语句使得流程直接跳到循环控制条件的测试部分，然后决定循环是否继续进行。在 for 循环中，遇到 continue 后，跳过循环体中余下的语句，而去对 for 语句中的表达式 3 求值，然后进行表达式 2 的条件测试，最后根据表达式 2 的值来决定 for 循环是否执行。在循环体内，不论 continue 是作为何种语句中的语句成分，都将按上述功能执行，这点与 break 有所不同。

注意 执行 continue 语句并没有使整个循环终止，而是结束本次循环，继续判断循环条件。

练一练

【练习 3-6】 输出 101～200 之间的全部素数。所谓素数 m 是指，除 1 和 m 之外，不能被 2～m−1 之间的任何整数整除（前面讲过查找区间可以缩小为 2～$\sqrt{m}$）。

解：显然，只要设计出判断某数 m 是否是素数的算法，外面再套一个 for 循环即可。判断某数 m 是否是素数的算法：根据素数的定义，用 2～$\sqrt{m}$ 之间的每一个数去整除 m，如果都不能被整除，则表示该数是一个素数。判断一个数是否能被另一个数整除，可通过判断它们整除的余数是否为 0 来实现。

解法程序如下：

```
#include  <math.h>
main()
{  int  m,n,i=0;
   for(m=101;m<=200;m+=2)          /*100 到 200 之间的偶数不能为素数*/
   {  for(n=2;n<=sqrt(m);n++)
         if(m%n==0)
            break;
      if(n>sqrt(m))
      {  printf("%5d",m);
         i++;
         if(i%10==0)
            printf("\n");
      }
   }
}
```

程序运行结果如下：

```
  101  103  107  109  113  127  131  137  139  149
  151  157  163  167  173  179  181  191  193  197
  199
```

【练习 3-7】 输出 100～200 之间不能被 3 整除的数。

解：设两个整型变量 n 和 k，n 为循环变量，k 为格式控制变量。令 n 的初值为 100，n 小于等于 200 的时候，判断 n 是否能被 3 整除，能整除，则提前结束本次循环。如果不能整除，则输出该值，并将 k 自加 1。当 k 整除 10 时，输出一个换行符。

解法程序如下：

```
main()
{  int  n,k=0;
    for(n=100;n<=200;n++)
    {  if(n%3==0)
          continue;
       printf("%5d",n);
       k++;
       if(k%10==0)
          printf("\n");
    }
}
```

程序运行结果如下：

```
  100  101  103  104  106  107  109  110  112  113
  115  116  118  119  121  122  124  125  127  128
  130  131  133  134  136  137  139  140  142  143
  145  146  148  149  151  152  154  155  157  158
  160  161  163  164  166  167  169  170  172  173
  175  176  178  179  181  182  184  185  187  188
  190  191  193  194  196  197  199  200
```

【练习 3-8】 百钱百鸡问题：假设 1 只公鸡卖 5 文钱，1 只母鸡卖 3 文钱，3 只小鸡卖 1 文钱，如果用 100 文钱买 100 只鸡，问公鸡、母鸡和小鸡各占多少只？

解：因为一共有 100 只鸡，所以假设公鸡、母鸡和小鸡的数量为 i、j 和 k，则有等式 i+j+k==100 成立。又因为一共是 100 文钱买 100 只鸡，所以每种鸡的数量乘以其花的钱数之和为 100 文钱，即有 5*i+3*j+k/3==100 成立。又因为 C 语言中的“/”两侧都是整数时，运算为整除，结果不准确，而 3 只小鸡卖 1 文钱，所以小

鸡的数量 k 应该是 3 的倍数，所以有 k%3==0 成立。这三个等式同时成立时的 i、j 和 k 值即为符合条件的三种鸡的数量。

源程序如下：

```
main()
{  int i,j,k;
   clrscr();              /*清屏*/
   for(i=1;i<=20;i++)
    for(j=1;j<=33;j++)
      for(k=1;k<=100;k++)
        if(i+j+k==100 && 5*i+3*j+k/3==100 && k%3==0)
           printf("\ncock:%2d,  hen:%2d,  chicken:%2d",i,j,k);
}
```

程序运行结果如下：

```
cock: 0,  hen:25,  chicken:75
cock: 4,  hen:18,  chicken:78
cock: 8,  hen:11,  chicken:81
cock:12,  hen: 4,  chicken:84
```

本讲小结

break 语句和 continue 语句主要用于循环的流程控制，二者在用法上有显著差别。break 语句用于结束其所在的 switch 分支结构或循环结构。continue 语句用于结束本次循环。

本章小结

本章主要介绍了选择结构和循环结构程序设计的语法结构、使用方法及注意事项，以及与之相关的流程控制语句。通过本章学习，应熟练掌握选择结构中 if 语句、switch 语句的意义和使用方法，熟练掌握 for 循环、while 循环和 do-while 循环的基本结构和应用，掌握 break 和 continue 语句的使用方法及流程转向语句 goto 语句的适用范围及用法。

课后习题三

一、选择题

1．以下程序的输出结果是_____。

A）0　　B）1　　C）2　　D）3

```
main( )
 {  int  a=2,b=-1,c=2;
    if (a<b)
      if(b<0)  c=0;
    else    c+=1;
    printf("%d\n",c);
}
```

2．以下程序的输出结果是_____。

A）1　　B）2　　C）3　　D）4

```
main()
{  int  w=4,x=3,y=2,z=1;
   printf ("%d\n",(w<x?w:z<y?z:x));
}
```

3．若执行以下程序时从键盘输入 3 和 4，则输出结果是_____。

A）14　　B）16　　C）18　　D）20

```
main()
{  int  a,b,s;
    scanf("%d%d",&a,&b);
    s=a;
    if(a<b)  s=b;
    s*=s;
    printf("%d\n",s);
}
```

4．运行以下程序后，输出_____。

A）****　　B）&&&&　　C）####&&&&　　D）有语法错误不能通过编辑

```
main( )
{  int  k=-3;
    if(k<=0)  printf("****\n");
    else     printf("&&&&\n");
}
```

5．以下程序段的输出结果是_____。

A）9　　B）6　　C）11　　D）10

```
main()
{  int  k,j,s;
    for(k=2;k<6;k++)
    {  s=1;
       for(j=k;j<6;j++)
           s+=j;
}
printf("%d\n",s);
}
```

6．以下程序段的输出结果是_____。

A）12　　B）15　　C）20　　D）25

```
int  i,j,m=0;
for(i=1;i<=15;i+=4)
    for(j=3;j<=19;j+=4)
       m++;
printf("%d\n",m);
```

7．以下程序段的输出结果是_____。

A）1　　B）3　0　　C）1　−2　　D）死循环

```
int  x=3;
do
{  printf("%3d",x-=2);
}while(!(--x));
```

8．以下程序段的输出结果是_____。

A）15　　B）14　　C）不确定　　D）0

```
main()
{  int  i,sum=0;
   for(i=1;i<6;i++ )
      sum+=i;
   printf("%d\n",sum);
}
```

9．以下程序段的输出结果是_____。

A）*#*#*#$　　B）#*#*#$　　C）*#*#$　　D）#*#*$

```
main()
{  int  i;
   for(i=1;i<=5;i++)
   {  if (i%2)  printf("*");
      else      continue;
      printf("#");
  }
    printf("$\n");
}
```

10．以下叙述正确的是_____。

A）do-while 语句构成的循环不能用其他语句构成的循环来代替。

B）do-while 语句构成的循环只能用 break 语句退出。

C）用 do-while 语句构成循环时，只有在 while 后的表达式为非零时结束循环。

D）用 do-while 语句构成循环时，只有在 while 后的表达式为零时结束循环。

11．以下程序段的输出结果是_____。

A）39　81　　B）42　84　　C）26　68　　D）28　70

```
main()
{  int    x,   i;
for(i=1;i<=100;i++)
{  x=i;
   if(++x%2==0)
     if(++x%3==0)
       if(++x%7==0)
         printf("%d  ",x);
}
}
```

12．若变量已正确定义，有以下程序段：

```
i=0;
do
{  printf("%d,",i);
}while(i++);
printf("%d\n",i);
```

其输出结果是_____。

A）0,0　　B）0,1　　C）1,1　　D）程序进入“死循环”状态

13．以下 for 循环的执行次数是_____。

```
for( x=0,y=0;(y!=123)&&(x<4);x++);
```

A）是无限循环　　B）循环次数不定　　C）4 次　　D）3 次

14．下面程序段运行结果是_____。

A）a=3，b=11　　B）a=2，b=8　　C）a=1，b=−1　　D）a=4，b=9

```
main()
{  int  a=1,b=10;
   do
   {  b-=a; a++;
   }while(b--<0);
   printf("a=%d,b=%d\n",a,b);
}
```

15. 下面程序段运行结果是_____。

A）1　　B）1　　C）1　　D）1

2　　2　　2

3　　3

4

```
main()
{  int  num=0;
   while(num<=2)
   {  num++;
      printf("%d\n",num);
   }
}
```

二、填空题

1. 以下程序的输出结果是_____。

```
main( )
{   int  a=100;
   if(a>100)   printf("%d\n",a>100);
   else        printf("%d\n",a<=100);
}
```

2. 当 a=1,b=2,c=3 时，以下 if 语句执行后，a、b、c 中的值分别为_____。

```
if(a>c)
   b=a; a=c; c=b;
```

3. 当执行以下程序段后，i 的值是_____，j 的值是_____，k 的值是_____。

```
int  a,b,c,d,i,j,k;
a=10; b=c=d=5; i=j=k=0;
for(;a>b;++b)  i++;
while(a>++c)   j++;
do
{  k++;
}while(a>d++);
```

4. 以下程序段的输出结果是_____。

```
int  k,n,m;
n=10; m=1; k=1;
while(k<=n)
{ m*=2;k++;}
printf("%d\n",m);
```

5. 以下程序段的输出结果是_____。

```
main( )
{  int  x=2;
   while(x--);
     printf("%d\n",x);
```

```
}
```

6．以下程序段的输出结果是_____。

```
int  i=0, sum=1;
do
{  sum+=i++;
}while(i<5);
printf("%d\n",sum);
```

三、编程题

1．编写程序，输入一个整数，打印出它是奇数还是偶数。

2．输出 1～20 中能被 5 整除的数，并求出它们的和。

3．输入任意一个整数，将其逆序输出，例如输入 1234，输出 4321。

第四章　数　　组

在前面各章中，我们已经学习了 C 语言所提供的简单数据类型，使用这些数据类型可以描述并处理一些简单的问题。然而实际需要处理的数据常常不止一个，如果仍然用基本数据类型来进行处理，就很麻烦且容易出错。例如，处理学生学习成绩时，就可以使用数组解决这个问题。

数组是 C 语言提供的一种最简单的构造类型，是一组具有相同数据类型的数据的有序集合。

第八讲　一维数组



学习目标

- 掌握一维数组的定义；
- 掌握一维数组的初始化；
- 掌握一维数组的引用；
- 掌握一维数组的输入、输出方法。

【思考题 4-1】　已知 10 个整数，求其最大值并显示在屏幕上。



程序分析

以前我们所学习的数据类型都是单一数据类型（如整型、浮点型、字符型等）。这里要求的 10 个整数，也可以用整型类型来分别定义各个变量，但需要大量不同的标识符作为变量名，并且这些变量在内存中的存放是随机的，如果想存储多个数据，组织和管理好这些变量会使程序变得复杂。

C 语言提供了一种最简单的构造类型——数组。在这种数据类型中存储的是相同类型的数据，我们可以直接定义一个整型数组并初始化该数组，利用下标值设计循环来找出其中的最大值，这样可以缩短和简化程序。下面通过一个例程来了解一维数组的定义和其使用方法。



编写程序代码

```
main()
{  int  i,max;
   int  a[10]={9,25,4,16,8,1,0,100,32,7};     /*定义数组 a，并初始化*/
   max=a[0];
   for(i=1;i<10;i++)
   {  if(a[i]>max)
         max=a[i];
   }
   printf("max=%d\n",max);
}
```



调试运行程序

程序运行结果如下：

```
max=100
_
```



1．一维数组的定义

数组是一组具有相同数据类型的数据的有序集合，其中的每个数据称为一个元素。数据元素在内存中占有连续的内存单元，不同的元素由一个统一的数组名和不同的下标来唯一标识。其定义格式如下：

```
存储类型符  类型说明符  数组名[常量表达式];
```

其中存储类型符用来说明数据的存储类型，默认为自动类型（存储类型的详细内容在第五章中讲述）。类型说明符为数组元素的数据类型，可以是 int 型、float 型、char 型以及后面章节将要讲到的指针、结构体和共用体等各种复合数据类型。数组名的定义符合 C 语言的标识符定义即可（即与普通变量名定义要求相同）。中括号“[]”为数组定义的分界符号，中括号里面为一维数组的元素个数。常量表达式必须正整型，表示定义数组中的元素个数。

在定义一个数组时，应注意以下几个问题。

（1）数组名规定必须遵守标识符命名规则，同一函数中的数组名不能与其他变量同名。例如下面的定义是错误的。

```
main()
{  float  a;
   int  a[5];
     …
}
```

在 main 函数中定义了一个 float 型变量 a，又定义了一个 int 型数组 a，二者同名，这是不合法的。

（2）[] 表示定义的是数组变量，不能省略。其中的整型常量表达式表示数组中元素个数，即数组的长度。

（3）常量表达式中可以包括常量和符号常量，不能包含变量。C 语言不允许对数组做动态定义。例如：

```
main()
{  int  a;
   scanf("%d",&a);
   int  b[a];
   …
}
```

这里数组 b 的长度由变量 a 来说明，是不合法的，因为在编译时，C 编译器是根据确定的数组大小来分配内存的。

例如：

```
#define  N  10
main()
{  int  b[N];
    …
}
```

数组 b 的长度由符号常量 N 来说明，这是合法的。

（4）数组元素的下标从 0 开始，称为下标的下界；最后一个元素的下标为元素个数减 1，称为下标的上界。



2．一维数组的初始化

一旦定义了数组，编译程序就为数组在内存中开辟了一段连续的存储单元，其首地址由数组名标识，数组的各个元素按顺序依次存放。最初，这些存储单元中没有确定的值，需对其进行初始化。

对数组元素的初始化可以用以下方法实现。

（1）在定义数组时对数组元素赋以初值，例如：

```
int  a[10]={0,1,2,3,4,5,6,7,8,9};
```

将数组元素的初值按顺序放在一对花括号内，初值类型必须与数组元素的类型一致，各初值之间用逗号分隔。初始化列表中的值依次从数组下标为 0 的元素开始赋值，不能跳过前面的元素而给后面的元素赋初值。

（2）可以只给一部分元素赋初值，例如：

```
int  a[10]={0,1,2,3,4};
```

数组中有 10 个元素，但初始化列表中只有 5 个值，此时只给数组 a 前 5 个元素赋初值，后 5 个元素的初值默认为 0。若数组的数据类型为字符型，则未指定值的数组元素值默认为空字符。当初始化列表中初值的个数多于数组元素的个数时，编译时将出错。

（3）如果想使一个数组中全部元素值为 0，可以写成：

```
int  a[10]={0,0,0,0,0,0,0,0,0,0};
```

或

```
int  a[10]={0};
```

不能写成:

```
int  a[10]={0*10};
```

（4）在给全部数组元素赋初值时，可以不指定数组长度，例如：

```
int  a[5]={1,2,3,4,5};
```

可以写成：

```
int  a[]={1,2,3,4,5};
```

编译时程序会根据初始化列表中初值的个数自动定义数组长度。



注意 若初值个数与元素个数不同时，则必须指定数组长度。

C 语言除了在定义数组变量时用初值列表为数组整体赋值之外，不能在其他情况下对数组变量做整体赋值。



3．一维数组的引用

数组必须先定义后使用。C 语言规定只能逐个引用数组元素，而不能一次引用整个数组。

一维数组元素的引用格式为：

数组名[下标表达式];

在引用数组元素时，应注意以下几个问题。

（1）“下标表达式”可以是整型常量、整型变量或整型表达式，其值均为非负整型数据，下标值从 0 开始，合法取值范围是 0≤下标表达式≤元素个数-1。

注意 C 语言不自动检查下标是否超范围，故必须在设计阶段从程序逻辑上保证下标不超范围。

（2）一个数组元素，相当于一个同类型的简单变量，可以对它进行赋值和参与各种运算。

（3）数值数组作为一个整体，不能参加数据运算，只能对单个的元素进行处理。

小讲堂

4．一维数组的输入、输出方法

（1）指定各元素逐一输入、输出。

```
scanf("%d%d%d%d%d",&a[0],&a[1],&a[2],&a[3],&a[4]);
printf("%d%d%d%d%d\n",a[0],a[1],a[2],a[3],a[4]);
```

（2）利用循环控制下标变化实现输入、输出。

```
int  a[10],i;
for(i=0;i<10;i++)
   scanf("%d",&a[i]);
for(i=0;i<10;i++)
   printf("%d",a[i]);
```

练一练

【练习 4-1】 从键盘输入某学科竞赛组 6 名成员的成绩（整数），求平均成绩、最高分、最低分。

解：本例采用一维整型数组存储 6 名成员的成绩，程序要求从键盘中输入成绩，可以利用循环语句调用 scanf 函数逐个输入各数组元素的数据。分析该程序，将第 1 个人的成绩预置为最高分 max 和最低分 min，用其余 5 人的成绩依次与 max 和 min 比较：如果某人成绩大于 max，则更新 max；如果某人成绩小于 min，则更新 min。

源程序如下：

```
#define  N  6                        /*定义符号常量 N（分数个数）*/
main()
{  int  score[N],i,sum,max,min;
   /*输入 N 个成绩*/
   printf("Please input %d scores(departed by space):",N);
   for(i=0;i<N;i++)
      scanf("%d",&score[i]);
   /*求分数合计、最高分和最低分*/
   sum=max=min=score[0];           /*预置累计和、最高、最低分*/
   for(i=1;i<N;i++)
   {  sum+=score[i];               /*求累计和*/
      if(score[i]>max)   max=score[i];
      if(score[i]<min)   min=score[i];
   }
   /*输出结果*/
```

```
    printf("\n average=%4.1f,max=%d,min=%d\n",(float)sum/N,max,min);
}
```

程序运行结果如下：

```
Please input 6 scores(departed by space): 56 90 88 65 50 92
average = 73.5, max = 92, min = 50
```

【练习 4-2】 用起泡法对 6 个整数进行升序排列。

解：起泡法排序的思路是，首先对 n 个数的每相邻两个数进行比较，小数放在前面，大数放在后面，经过第一遍扫描后，数列的最后一个数就是最大数；接着对前 n−1 个数进行同样的比较，将次大数放在倒数第二位置上，依次类推，直到排序结束为止。在这个过程中，大数不断往下沉，小数不断往上冒，故称为起泡法排序。

本例中对 10，6，3，9，1，7 这 6 个数排序，则要进行 5 轮次的比较。排序过程如图 4-1 所示（括号内为已排好序的序列）。在第一轮次比较中要进行 5 次两两比较，在第二轮次比较中要进行 4 次两两比较。若有 n 个数，在每轮次比较中要进行 n−1 次两两比较，在第二轮次比较中要进行 n−2 次两两比较，在第 i 轮次比较中要进行 n−i 次两两比较。

初始状态：	10	6	3	9	1	7
第一趟排序结果：	6	3	9	1	7	(10)
第二趟排序结果：	3	6	1	7	(9	10)
第三趟排序结果：	3	1	6	(7	9	10)
第四趟排序结果：	1	3	(6	7	9	10)
第五趟排序结果：	1	(3	6	7	9	10)

图 4-1 起泡排序法的排序过程

源程序如下：

```
#include  "stdio.h"
main()
{   int  i,j,temp,a[6];
    printf("please input 6 numbers:\n");
    for(i=0;i<6;i++)
       scanf("%d",&a[i]);
    printf("\n");
    for(j=1;j<=5;j++)              /*比较 5 轮次*/
      for(i=0;i<6-j;i++)           /*第 j 轮次比较 6-j 次*/
        if(a[i]>a[i+1])            /*比较相邻两数大小*/
        {  temp=a[i];
           a[i]=a[i+1];
           a[i+1]=temp;
        }
    printf("the sorted numbers:\n");
    for(i=0;i<6;i++)
      printf("%d  ",a[i]);
}
```

程序运行结果如下：

```
please input 6 numbers:
10 6 3 9 1 7

the sorted numbers:
1  3  6  7  9  10
```

我们可以对上例中的程序做一些小的修改，通过一个计数器和两重循环将每趟排序的结果显示出来，使排序更明显。

源程序如下：

```
#include "stdio.h"
main()
{  int  i,j,temp,a[6];
   int  count=0;                          /*计数器，记录是第几趟排序*/
   printf("please input 6 numbers:\n");
   for(i=0;i<6;i++)
      scanf("%d",&a[i]);
   printf("\n");
   for(j=1;j<=5;j++)
   {  count++;                            /*每排序一趟，计数器加 1*/
      for(i=0;i<6-j;i++)
         if(a[i]>a[i+1])
         {  temp=a[i];
            a[i]=a[i+1];
            a[i+1]=temp;
         }
      printf("%3d:",count);               /*打印是第几趟排序*/
      for(i=0;i<6;i++)
         printf("%d ",a[i]);              /*打印本趟排序的结果*/
      printf("\n");
   }
   printf("the sorted numbers:\n");
   for(i=0;i<6;i++)
      printf("%d  ",a[i]);                /*打印最终排序的结果*/
}
```

程序运行结果如下：

```
please input 6 numbers:
10 6 3 9 1 7

  1:6 3 9 1 7 10
  2:3 6 1 7 9 10
  3:3 1 6 7 9 10
  4:1 3 6 7 9 10
  5:1 3 6 7 9 10
the sorted numbers:
1  3  6  7  9  10
```

本讲小结

数组类型是 C 语言中构造类型数据的一种，数组按照其下标的维数，可以分为一维数组、二维数组、多维数组。将数组与循环结合起来，可以有效地处理大批量的数据，大大提高工作效率，十分方便。

想一想

在【练习 4-2】中排序进行到第 4 趟就已经排序完毕。但实际上程序并没有就此结束，而是一直运行到循环结束，第 5 趟排序是空比较，浪费运行时间，我们应如何修改程序节省这部分时间呢？（提示：可以设

置一个标志，一旦发现在某趟无须交换数据，就提前跳出循环，终止排序）。试改写该程序并调试运行。

第九讲　二维数组

学习目标

- 掌握二维数组的定义;
- 掌握二维数组的初始化;
- 掌握二维数组的引用。

【思考题 4-2】　有一个 3×4 的矩阵，求其中的最大值，以及其所在的行号和列号。

程序分析

（1）定义二维数组 int a[3][4]，定义变量 max 存放最大值，row 存放最大值行号，col 存放最大值列号；

（2）将第一个元素 a[0][0]作为临时最大值 max，row 和 col 的初值设为 0；

（3）将 max 与每一个元素 a[i][j]进行比较，若 a[i][j]>max，把 a[i][j]作为新的临时最大值，并记录下其下标 i 和 j；

（4）当全部元素比较完后，max 是整个矩阵全部元素的最大值，输出最大值和其所在的行、列号。

编写程序代码

```
#include "stdio.h"
main()
{  int  i,j,row=0,col=0,max;
   int  a[3][4]={{1,2,3,4},{4,5,6,7},{7,8,9,0}};
   max=a[0][0];
   for(i=0;i<=2;i++)
     for(j=0;j<=3;j++)
       if(a[i][j]>max)
       {  max=a[i][j];
          row=i;
          col=j;
       }
   printf("max=%d,row=%d,clo=%d\n",max,row,col);
}
```

调试运行程序

程序运行结果如下：

```
max=9,row=2,col=2
_
```

小讲堂

1．二维数组的定义

当数组元素的下标有两个或两个以上时，称该数组为多维数组。下标为两个时，称为二维数组；下标为三个时，称为三维数组。多维数组中使用最多是二维数组，本节主要介绍二

维数组，其定义格式如下：

```
存储类型符　类型符　数组名[常量表达式 1][常量表达式 2];
```

其中常量表达式 1 是数组元素的行数，常量表达式 2 是数组元素的列数。与一维数组相比，二维数组的定义，除了增加了一个[常量表达式 2]外，其他都一样。

例如：

```
int  a[2][3];
```

定义了一个具有 2 行 3 列的二维整形数组 a。该数组共有 2×3 个元素，分别为：

```
a[0][0],a[0][1],a[0][2]
a[1][0],a[1][1],a[1][2]
```

C 语言对二维数组采用这样的定义方式，可以把二维数组看做是一种特殊的一维数组，它的元素又是一个一维数组。例如可以把数组 a 看做是一个一维数组，它有 2 个元素：a[0]和 a[1]，每个元素又是一个包含 3 个元素的一维数组。可以把 a[0]、a[1]看做是 2 个一维数组的名字。与一维数组一样，二维数组下标值也是从 0 开始的，二维数组元素在内存中的排列顺序为“按行存放”，占有一块连续的内存单元，即在内存中先存放第 0 行各列元素，再存放第 1 行各列元素，依次类推。

从二维数组中各元素在内存中的排列顺序可以计算出数组元素在数组中的排列位置。例如有一个 m×n 的二维数组 a，其中 i 行、j 列元素 a[i][j]在数组中的排列位置为：

```
i×n+j+1
```

（假设 a[0][0]位于数组的第一个位置）



2．二维数组的初始化

对数组元素的初始化可以用如下方法实现。

（1）初值按行的顺序依次排列，每行都用花括号括起来，各行之间用逗号隔开如：

```
int a[2][3]={{1,2,3},{4,5,6}};
```

第一个花括号内的数据依次赋给第一行的元素，第二个花括号内的数据依次赋给第二行的元素，依次类推。

（2）不分行的初始化。将所有元素的初值写在花括号内，按数组排列的顺序依次对各元素赋初值，如：

```
int a[2][3]={1,2,3,4,5,6};
```

如果初始化列表中元素的个数小于数组中元素的个数，则剩余元素赋默认值；若多于数组中元素的个数，则在编译过程中提示出错。

（3）对二维数组中部分元素初始化，如：

```
int a[2][3]={1,2},{4}};
```

（4）如果对二维数组的全部元素都初始化，可以省略第一维的定义，但不能省略第二维的定义。程序会自动计算初始化元素个数，并用元素个数除以二维数组的列数来求出该数组的行数，所以初始化时只能省略二维数组的行数，而列数不能省略。例如下面的两个定义是等价的：

```
int  a[ ][4]={1,2,3,4,5,6,7,8,9,10,11,12};
int  a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

（5）如果只对部分数组元素提供初值，又省略了第一维的长度，那么应分行赋初值，即

使某行没有对应的初值，也必须保留对应该行的花括号，如：

```
int a[ ][4]={{1,3},{},{5,6,7},{8,9,10,11}};
```

与一维数组特性相同，在定义变量时除了可以对二维数组赋初值外，不能用 C 语句对二维数组作整体赋值，只能利用循环语句对数组元素逐一赋值。



3．二维数组的引用

和一维数组元素的引用一样，二维数组元素也是通过数组名和下标来引用的，只是这里需要两个下标。

二维数组元素引用的格式为：

```
数组名[行下标表达式][列下标表达式];
```

在引用二维数组元素时需注意以下几个问题。

（1）下标同一维数组一样，可以是整型常量或是整型表达式。行下标表达式的取值范围为 0～行数−1，列下标表达式的取值范围为 0～列数−1。例如一个 3 行 4 列的二维数组其行下标最大值为 2，列下标最大值为 3。

（2）对基本数据类型的变量所能进行的各种操作，也都适合于同类型的二维数组元素。

（3）要引用二维数组的全部元素，就要遍历二维数组，通常应使用二层嵌套的 for 循环：一般常把二维数组的行下标作为外循环的控制变量，把列下标作为内循环的控制变量。

练一练

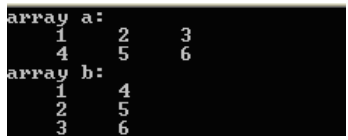
【练习 4-3】 将一个矩阵 A（2×3）转换成为其转置矩阵 A′（3×2）输出。

解：矩阵 A 的转置矩阵 A′是指矩阵 A 的第 0 行、第 1 行各元素，转换成转置矩阵 A′的第 0 列和第 1 列的各元素。

程序代码如下：

```
#include <stdio.h>
main()
{  int  a[2][3]={{1,2,3},{4,5,6}};
   int  b[3][2],i,j;
   printf("array a:\n");
   for(i=0;i<=1;i++)
   {  for(j=0;j<=2;j++)
      {  printf("%5d",a[i][j]);
         b[j][i]=a[i][j];
      }
      printf("\n");
   }
   printf("array b:\n");
   for(i=0;i<=2;i++)
   {  for(j=0;j<=1;j++)
        printf("%5d",b[i][j]);
      printf("\n");
   }
}
```

程序运行结果如下：



【练习 4-4】 输出以下的杨辉三角形（要求输出 10 行）。

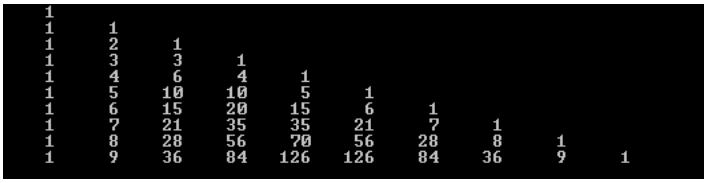
```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
…      …    …
```

解：杨辉三角形可以看做 N×N 方阵的下三角，其中第 0 列和对角线元素值为 1，其余各元素是上一行同列和上一行前一列的两个元素之和。

程序代码如下：

```
#define  N  10
main()
{  int  i,j,a[N][N];
   for(i=0;i<N;i++)
   {  a[i][i]=1;
      a[i][0]=1;
   }
   for(i=2;i<N;i++)
   {  for(j=1;j<i;j++)
      a[i][j]=a[i-1][j-1]+a[i-1][j];
   }
   for(i=0;i<N;i++)
   {  for(j=0;j<=i;j++)
        printf("%6d",a[i][j]);
      printf("\n");
  }
}
```

程序运行结果如下：



本讲小结

本讲主要介绍了二维数组的定义、初始化以及引用，二维数组中元素排列的顺序是按行存放的，即在内存中先顺序存放第一行的元素，再存放第二行的元素。二维数组可被看做是一种特殊的一维数组，它的元素又是一个一维数组。

想一想

本讲中介绍的二维数组各元素的成员都是数值型的，在实际应用中，经常需要处理字符及字符串，如果需要处理的对象是字型的数据，又应该怎样定义和使用呢？

第十讲　字符数组与字符串

学习目标

- 掌握字符数组的定义与初始化;
- 掌握字符数组的整体操作;
- 了解常用字符串处理函数。

【思考题 4-3】 从键盘输入一个字符串（字符串中不包含空格），当输入回车时认为输入结束。统计输入字符串中的小写英文字母、大写英文字母、数字字符、其他字符的个数。

程序分析

定义整型变量 m、n、x、y 分别存储统计得到的输入字符串中的小写英文字母、大写英文字母、数字字符、其他字符的个数。用一个单循环控制，对字符数组 s 中的元素逐个进行判断，根据不同情况（元素值是否为小写英文字母、大写英文字母、数字字符、其他字符）对相应计数变量计数。循环控制条件是当前所处理的字符串中的字符不是字符串结束标志'\0'，即一个字符串的所有字符还没有处理完毕。

编写程序代码

```
#include <stdio.h>
#define N 100
main()
{ int i,m,n,x,y;
  char s[N];
  printf("input a string:\n");
  scanf("%s",s);
  m=n=x=y=i=0;
  while(s[i]!='\0')
  { if(s[i]>='a'&&s[i]<='z')
      m++;
    else if(s[i]>='A'&&s[i]<='Z')
      n++;
    else if(s[i]>='0'&&s[i]<='9')
      x++;
    else
      y++;
    i++;
  }
  printf("the number of capital letters:%d\n",n);
  printf("the number of lowercase letters:%d\n",m);
  printf("the number of figures:%d\n",x);
  printf("other characters:%d\n",y);
}
```


调试运行程序

程序运行结果如下：

```
input a string:
asdGHJ123gdL@#!sd
the number of capital letters:4
the number of lowercase letters:7
the number of figures:3
other characters:3
```



1．字符串及其结束标志'\0'

前面的章节中我们接触过字符串常量的概念。字符串是指若干有效字符的序列，通常它用英文双引号括起来，也称为字符串常量。有效字符包括字母、数字、专用字符和转义字符等。在 C 语言中没有字符串变量，字符串不是存放在一个变量中，而是存放在一个字符型的数组中。C 语言规定用'\0'作为字符串的结束标志，当把一个字符串存入一个数组时，系统自动将结束符'\0'存入数组，以此作为字符串是否结束的标志。

'\0'代表 ASCII 码为 0 的字符，表示一个“空操作”，只起一个标志作用。它占内存空间，但不计入串的长度。因此可以对字符数组采用另一种方式进行操作——字符数组的整体操作。

有了结束标志'\0'以后，字符数组的长度就显得不那么重要了。在程序中往往根据'\0'来判定字符串是否结束，而不是根据数组的长度来决定字符串长度。在定义字符数组时需使数组长度始终大于字符串的实际长度。同时应注意的是，如果一个字符数组先后存放多个不同的字符串，则数组长度需大于最长的字符串的长度。

注意 由于系统在存储字符串常量时，会在串尾自动加上 1 个字符串结束标志'\0'，所以无须人为地再加 1 个。

另外，由于结束标志也要在字符数组中占用一个元素的存储空间，因此在定义字符数组时，其长度至少为所存储的字符串长度加 1。



2．字符数组的定义

字符数组是存放字符型数据的数组，字符数组的一个元素存放一个字符。字符数组的定义方法与定义其他类型数组的方法完全相同，但其类型必须为 char 型。

字符数组包括一维字符数组和二维字符数组等。其中一维字符数组用于存储和处理一个字符串，其定义格式与一维数值数组一样；二维字符数组用于同时存储和处理多个字符串，其定义格式与二维数值数组一样。

一维字符数组的定义格式为：

```
char  数组名[常量表达式];
```

其中常量表达式的值，就是一维字符数组元素的个数。

二维字符数组的定义格式为：

```
char  数组名[常量表达式 1][常量表达式 2];
```

其中常量表达式 1、常量表达式 2 的值，就是二维字符数组的行数和列数。



3．字符数组的初始化：

字符数组的初始化有两种方法。

（1）使用初始化列表，逐个为数组中各元素赋值。例如：

```
char  str[11]={'I',' ','a','m',' ','a',' ','b','o','y'};
```

若初始化列表中初值个数大于数组的长度，则按语法错误处理；若小于数组的长度，则只将这些字符赋给数组中前面那些元素，其余的元素自动定为空字符，即'\0'；若等于数组长度，定义字符数组时其长度可省略，系统会根据初值的个数确定字符数组的长度。

（2）直接将一个字符串常量赋给一个字符数组。例如：

```
char  str[]={"I am a boy"};
```



字符串的输入和输出函数存放在“stdio.h”头文件中，其他字符串处理函数存放在“string.h”头文件中，注意使用时在源程序开始处要包含相应的头文件。

4．字符串的输入和输出函数

1）常用的字符串输入函数 scanf 函数和 gets 函数

（1）scanf 函数。例如：有一个字符数组：

```
char  c[10];
```

可以利用 scanf 函数向其输入字符或字符串，输入一个字符并将其存到数组元素 c[0]中，其方法与简单字符变量的输入相同，即：

```
scanf("%c",&c[0]);
```

将一个字符串存入字符数组 c 中，即：

```
scanf("%s",c);
```

scanf 函数在输入字符串时，使用“%s”格式控制符，并且与“%s”对应的地址列表是一个字符数组名，当输入一个字符串后，scanf 函数会自动在字符串后面加上'\0'。由于数组名代表数组的起始地址，因此在 scanf 函数中只需写数组 c 的名字即可，不必加“&”。

使用 scanf 函数向数组输入字符串时必须注意输入的字符串中不能包含空格，因为 C 语言规定用 scanf 函数输入字符串时，以空格或回车符作为字符串间隔的符号。例如：

```
char  str1[10],str2[10],str3[10];
scanf("%s%s%s",str1,str2,str3);
```

输入数据：

```
We  love  china!
```

则 str1 中的字符串是“We”，str2 中的字符串是“love”，str3 中的字符串是“china!”。由上例可以看出我们无法利用 scanf 函数输入一个包含空格的字符串并将其赋值给一个字符数组。

（2）gets 函数。gets 函数能输入完整的句子，以回车键作为字符串结束符号，将从键盘输入的字符串存放到字符数组中。函数的调用格式为：

```
gets(字符数组名);
```

例如：

```
char  str[30];
gets(str);
```

输入一个字符串，以回车符作为本行结束符：

```
We love china!
```

则字符数组 str 中的字符串是“We love china!”。



注意 在接受字符串输入时，存放字符串的字符数组变量要足够大，否则容易引发数组越界操作，导致程序错误。

2）字符串的输出函数 printf 函数和 puts 函数

（1）printf 函数。printf 函数在输出字符串时使用“%s”格式控制符，并且与“%s”对应的输出项必须是要输出字符串的第一个字符的地址。printf 函数将依次输出字符串中的每个字符，直到遇到字符结束符'\0'。在字符串中可以有空格字符（这与用 scanf 函数不能输入带空格的字符串不同）。

例如：

```
char  str[]="We love china";
printf("%s",str);
```

输出结果是：

```
We love china
```

（2）puts 函数。puts 函数为字符串输出函数，能将一个字符串输出到终端，输出的字符串包含转义字符。函数的调用格式为：

```
puts(字符数组名);
```

例如：

```
char str[]="We love China";
```

输出结果是：

```
We love China
```

输出字符串时，字符串结束标志'\0'转换为换行符'\n'，即输出字符串后换行。

字符串的输入/输出要注意下面几个问题。

① 输出字符串内容中不包括结束标志符'\0'。

② 用%s 格式输入或输出字符数组时，函数 scanf 的地址项、函数 printf 的输出项都是字符数组名，而不是数组元素名。这时数组名前不能再加“&”符号，因为数组名就是数组的起始地址，也不能加下标。

③ 如果字符数组长度大于字符串实际长度，在按整个字符串输出时，只输出到遇'\0'结束，即'\0'以后的内容将不会输出。

④ 如果一个字符数组中包含一个以上'\0'，则遇第一个'\0'时输出就结束。

⑤ 用语句 scanf("%s",s)；为字符数组 s 输入数据时，遇空格键或回车键时结束输入，但所读入的字符串中不包含空格键或回车键，而是在字符串末尾添加'\0'。

⑥ 用 printf 函数以格式符%s 输出字符串时，首先按字符数组名找到其数组的起始地址，然后从输出项提供的地址开始输出，逐个输出其中的字符，直到遇字符串结束符'\0'为止。

⑦ 调用 gets 函数和 puts 函数的源程序文件中要包含 stdio.h 头文件。



5．常用字符串处理函数

1）字符串连接函数 strcat

调用格式为：

```
strcat(字符数组名 1,字符数组名 2);
```

功能：连接两个字符数组中的字符串，把字符串 2 接到字符串 1 的后面，并删除字符数组 1 中字符串后的结束标志'\0'。结果放在字符串 1 中，函数调用后得到函数值——字符数组 1 的地址。例如：

```
char s1[20]="Olympic ",s2[10]="Sports";
strcat(s1,s2);
printf("%s",s1);
```

输出结果：

```
Olympic Sports
```



在调用 strcat 函数时，必须确保字符数组 1 的定义长度一定得足够容纳字符串 1 和字符串 2 的长度之和。

2）字符串复制函数 strcpy

调用格式为：

```
strcpy(字符数组名 1,字符串 2);
```

功能：将字符串 2 拷贝到字符数组 1 中去，字符串 2 的结束标志'\0'也一同拷贝。字符串 2 可以是字符串常量，也可以是字符数组名，这时相当于把一个字符串赋予一个字符数组。例如：

```
char s1[20],s2[]="Hello World!";
strcpy(s1,s2);
printf("%s",s1);
```

输出结果是：

```
Hello World!
```

使用 strcpy 函数时需注意以下几个问题：

- 字符数组 1 应有足够的长度，以便存入所拷贝的字符串；
- 字符串的复制必须使用 strcpy 函数，而不能使用赋值运算符“=”。

如果只把字符串 2 的一部分拷贝到字符串 1 可以使用 strncpy 函数，该函数的调用格式为：

```
strncpy(字符数组名 1,字符串 2,长度 n);
```

功能：将字符串 2 的前 n 个字符拷贝到字符数组 1 中，并在末尾加'\0'。例如：

```
char s1[20],s2[]="Hello World!";
strncpy(s1,s2,3);
printf("%s",s1);
```

输出结果是：

```
Hel
```

3）字符串比较函数 strcmp

调用格式为：

```
strcmp(字符串 1,字符串 2);
```

功能：按 ASCII 码值大小比较，将两个字符串自左至右逐个字符相比较，直到出现不同的字符或到'\0'为止。如果全部字符相同，则认为相等；如果出现不相同的字符，则以第一个不相同的字符的比较结果为准。比较的结果由函数值带回。参数字符串 1 和字符串 2 可以是字符数组名，也可以是字符串常量。比较有几种情况：

- 字符串 1=字符串 2，返回值为 0。
- 字符串 1>字符串 2，返回值为大于 0 的整数。
- 字符串 1<字符串 2，返回值为小于 0 的整数。

例如：

```
int  i;
char  s1[20]="abc",s2[20]="ade";
i=strcmp(s1,s2);
printf("%d",i);
```

输出结果是：

```
-2
```

该函数的返回值为字符串中对应位置的第一个不同的两个字符的 ASII 码的差值，所以上例中的第一个不相同的字符为'b'和'd'，这两个字符的 ASCII 码分别为 98 和 100，所以差值为−2，函数返回−2。

注意 字符串比较大小时只能用 strcmp 函数，而不能用关系运算符“==”。

4）求字符串长度函数 strlen

调用格式为：

```
strlen(字符数组);
```

功能：求字符串长度。函数值为字符串的实际长度，不包括'\0'在内。例如：

```
char s1[]="Hello World!";
printf("The length of the string is :%d\n",strlen(s1));
```

输出结果是：

```
The length of the string is :12
```

5）字符串小写函数 strlwr

调用格式为：

```
strlwr(字符串);
```

功能：将字符串中的大写字母转换成小写字母，其余字符不变。例如：

```
char s1[]="Hello World!";
printf("%s\n",strlwr(s1));
```

输出结果是：

```
hello world!
```

6）字符串大写函数 strupr

调用格式：

```
strupr(字符串);
```

功能：将字符串中的小写字母转换成大写字母，其余字符不变。例如：

```
char s1[]="Hello World!";
printf("%s\n",strupr(s1));
```

输出结果是：

```
HELLO WORLD!
```

练一练

【练习 4-5】 设计一个程序，将一个字符串逆序存放并显示。

解：字符串的逆序存放可以通过一个 for 循环实现，即将第一个字符与最后一个字符交换，第二个字符与倒数第二个字符交换，依次类推，一直到中间字符为止。

程序代码如下：

```
#include  "stdio.h"
#include  "string.h"
main()
{  char  c,str[40];
   int  i,length;
   printf("input string:");
   gets(str);
   length=strlen(str);
   for(i=0;i<length/2;i++)
   {  c=str[i];
      str[i]=str[length-i-1];
      str[length-i-1]=c;
   }
   printf("\noutput string:");
   puts(str);
}
```

程序运行结果如下：

```
input string:asdfg

output string:gfdsa
_
```

【练习 4-6】 有一行电文，已按规律译成密码，密码规则为第一个字母变成第 26 个字母，第 i 个字母变成第（26−i+1）个字母，非字母字符不变。要求编程序将密码译回原文，并打印出密码和原文。

解：由题意可知，程序设计步骤如下。

（1）输入字符串 ch。

（2）对字符串中的每个元素依次进行如下操作：判断该元素是否在 A～Z 之间，若该元素在 A～Z 之间，则该元素的值为 26−(ch[i]−64)+1+64；判断该元素是否在 a～z 之间，若该元素在 a～z 之间，则该元素的值为 26−(ch[i]−96)+1+96。

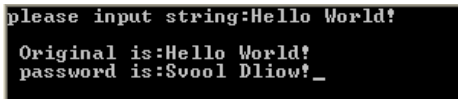
（3）当全部元素比较完后，输出其原值和转换后的值。

编写程序代码如下：

```
#include "stdio.h"
main()
```

```
{   int  i,n;
    char  ch[80],tran[80];
    printf("please input string:");
    gets(ch);
    printf("\n Original is:%s",ch);
    i=0;
    while(ch[i]!='\0')
    {  if((ch[i]>='A')&&(ch[i]<='Z'))
         tran[i]=26-(ch[i]-64)+1+64;
       else  if((ch[i]>='a')&&(ch[i]<='z'))
         tran[i]=26-(ch[i]-96)+1+96;
       else
         tran[i]=ch[i];
       i++;
    }
    n=i;
    printf("\n password is:");
    for(i=0;i<n;i++)
      putchar(tran[i]);
}
```

程序运行结果如下：



本讲小结

C 语言中没有字符串类型的变量，在处理字符串时通常采用字符数组的方法。本讲主要介绍了字符数组和字符串，字符数组逐个引用时其使用方法与数值数组相同，而其整体使用是字符数组特有的。

想一想

在输入多个字符串时，我们应该使用什么函数？如果想将两个字符串连接起来该使用什么函数？

本章小结

数组是程序设计中最常用的数据结构。它是一种构造类型，使用数组，可以将类型相同的相关数据连续存放。数组汇总的各个数据称为数组元素，不同元素用其在数组的位置（即下标）标识。

数组可分为一维数组或多维数组。本章主要介绍了一维数组和二维数组。在定义数组时，数组长度即元素个数必须是确定的，应该用常量来定义数组的长度而不能使用变量。数组定义由数组类型、数组名、数组长度三部分组成。数组元素又称为下标变量。数组类型是指数组元素的类型。定义数组时可以对其初始化，可只对部分元素初始化，也可以对全部元素初始化。当对全部元素初始化时，数组长度可以省略。

除了对字符串、字符串数组可以利用相应的字符串处理函数做整体运算外，对数组的任何操作只能对数组元素进行。

数组元素的引用采用下标法。在 C 语言中，下标的取值从 0 开始，上限为数组长度减 1。在使用时应注意下标不可越界。

字符串是特殊的一维字符数组，以字符串结束标志'\0'结尾。可以使用字符串处理函数对字符串进行操作，使用字符串处理函数时，应包含文件“string.h”。

课后习题四

一、选择题

1．在 C 语言中，引用数组元素时，其数组下标的数据类型是_____。

A）整型常量　　B）整型表达式

C）整型常量或整型表达式　　D）任何类型的表达式

2．以下对一维整型数组 a 的正确定义是_____。

A）int a(10);　　B）int n=10,a[n];

C）int n;　　D）#define SIZE 10

scanf("%d ",&n);　　int a[SIZE];

int a[n];

3．若有说明 int　a[10];，则对 a 数组元素的正确引用是_____。

A）a[10]　　B）a[3.5]　　C）a(5)　　D）a[10−10]

4．以下能对一维数组 a 进行正确初始化的语句是_____。

A）int a[10]=(0,0,0);　　B）int a[10]={ };

C）int a[]={0};　　D）int a[10]={10*1};

5．以下定义不正确的语句是_____ 。

A）double x[5]={2.0,9.0,8.9,0.0,1.5};　　B）int y[5]={1,2,3,4,5,6};

C）char c1[]={'1','2','3','4','5'};　　D）char c2[]={'\x10','\xa','\x8'};

6．以下对二维数组正确说明是：_____。

A）int　a[3][];　　B）float　a(3,4);　　C）double　a[1][4];　　D）float　a(3)(4);

7．若有说明 int　a[][3]={1,2,3,4,5,6,7};，则 a 数组第一维的大小是_____。

A）3　　B）2　　C）4　　D）无确定值

8．定义如下变量和数组：

```
int  k;
int  a[3][3]={1,2,3,4,5,6,7,8,9};
```

则下面语句的输出结果是_____。

```
for (k=0;k<3;k++)
printf("%d",a[k][2-k]);
```

A）3 5 7　　B）3 6 9　　C）1 5 9　　D）1 4 7

9．下面是对 s 的初始化，其中不正确的是_____。

A）char　s[5]={"abc"};　　B）char　s[5]={'a','b','c'};

C）char　s[5]= "a";　　D）char　s[5]="abcdef";

10．在 C 程序中使用字符串处理函数时，在#include 命令行中包括_____。

A）graphics.h　　B）ctype.h　　C）string.h　　D）math.h

11．表达式 strcmp("box","boss")的值是一个_____。

A）正数　　B）0　　C）负数　　D）不确定

12．有两个字符数组 a、b，则以下正确的输入语句是_____。

A）gets(a,b)；　　B）scanf("%s%s",a,b)；

C）scanf("%s%s",&a,&b)；　　D）gets("a"),gets("b")；

13．有如下定义：char　str[5]={ 'a','b','\0','c','\0'}；则语句 printf("%s",str);的输出结果是_____。

A）'a"b '　　B）ab　　C）abc　　D）ab c

14．下面叙述正确的是_____。

A）两个字符串所包含的字符个数相同时，才能比较字符串

B）字符个数多的字符串比字符个数少的字符串大

C）字符串"STOP　"与"STOP"相等

D）字符串"That"小于字符串"The"

15．下面对 C 语言字符数组描述中错误的是_____。

A）字符数组可以存放字符串

B）字符数组的字符串可以整体输入、输出

C）可以在赋值语句中通过赋值运算符“=”对字符数组整体赋值

D）不可以用关系运算符对字符数组中的字符串进行比较

二、填空题

1．在 C 语言中，二维数组元素在内存中的存放顺序是_____。

2．若有定义 double　x[4][5];，则 x 数组中行下标的下限为_____，列下标的上限为_____。

3．若有二维数组 a[3][4]，则元素 a[2][2]在数组中的位置为_____。（假设 a[0][0]位于数组的第一个位置上。）

4．若有定义 int a[3][4]={{1,2},{0},{4,6,8,10}};，则初始化后，a[1][2]得到的初值是_____。

5．以下程序段的输出结果是_____。

```
#include <stdio.h>
main()
{  char  str[30];
   strcpy(&str[0],"CH");
   strcpy(&str[1],"DEF");
   strcpy(&str[2],"ABC");
   puts(str);
}
```

6．以下程序段的输出结果是_____。

```
main()
{  char  str[]="Happy new year!";
   str[5]='\0';
   puts(str);
}
```

7．设有定义 char s[12]= "string";，则 printf("%d\n",strlen(s));的输出是_____。

8．下面程序可求出矩阵 a 的两条对角线上的元素之和，请填空。

```
main()
{  int  a[3][3]={1,3,6,7,9,11,14,15,17},sum1=0,sum2=0,i,j;
   for(i=0;i<3;i++)
     for(j=0;j<3;j++)
       if(i==j)
          sum1=sum1+a[i][j];
```

```
    for(i=0;i<3;i++)
      for(【1】;【2】;j--)
        if((i+j)==2)
           sum2=sum2+a[i][j];
   printf("sum1=%d,sum2=%d\n",sum1,sum2);  }
```

【1】_____【2】_____

9．当从键盘输入 16 并回车后，下面程序的运行结果是_____。

```
main()
{  int  x,y,i,a[8],j,u,v;
   scanf("%d",&x);
   y=x;i=0;
   do
   {  u=y/2;
      a[i]=y%2;
      i++;
      y=u;
    }while(y>=1);
    for(j=i-1;j>=0;j--)
       printf("%d",a[j]);
}
```

10．下面程序的运行结果是_____。

```
main()
{  int  i=1,n=3,j,k=3;
   int  a[5]={1,4,5};
   while(i<=n && k>a[i])
     i++;
   for(j=n-1;j>=i;j--)
     a[j+1]=a[j];
   a[i]=k;
   for(i=0;i<=n;i++)
     printf("%3d",a[i]);
}
```

三、编程题

1．试从键盘输入 5 个整数，找出最大数和最小数所在的位置，并把二者对调，然后输出调整后的 5 个数。

2．已知矩阵 a、b，求 a、b 的和，并将结果存入矩阵 c 中，试以矩阵的形式输出矩阵 c。（说明：矩阵 a 为 3 行 4 列，其值为{3,−2,7,5,1,0,4,−3,6,8,0,2}，矩阵 b 为 3 行 4 列，其值为{−2,0,1,4,5,−1,7,6,6,8,0,2}。

3．输入一行字符，统计其中单词的个数。（单词之间用空格分隔开）

第五章　函　　数

C 语言是通过函数来实现模块化程序设计的。较大的 C 语言应用程序往往是由多个函数组成的，每个函数分别对应各自的功能模块。从用户的使用角度看，函数有两种：标准函数（即库函数）和用户自定义函数。本章主要讨论的是用户自定义函数。通过本章的学习，读者应该掌握以下内容：

- 函数的定义与函数声明；
- 函数的调用；
- 函数的嵌套调用与递归调用；
- 数组作为函数参数；
- 内部变量与外部变量；
- 变量的作用域及其存储类型；
- 内部函数与外部函数。

第十一讲　函数定义、调用、函数原型及函数返回语句



学习目标

- 掌握函数的分类；
- 掌握函数的定义格式；
- 掌握函数的调用；
- 掌握函数的声明，能够区分函数定义与函数原型。

一、函数的定义、调用及函数返回语句

【思考题 5-1】　定义一个函数，用于求两个数中较大的数。



程序分析

（1）在本程序中定义两个函数：main 函数和 max 函数。

（2）max 函数中定义一个变量 c，存放两个参数中较大的数，通过 return 语句把 c 的值返回调用函数。

（3）主函数中通过调用 max 函数求两个数中较大的数。



编写程序代码

```
#include "stdio.h"
double max(double a,double b)
{ double c;
  if(a>b)
```

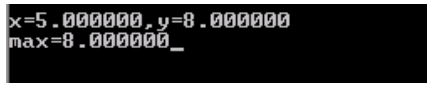
```
    c=a;
  else
    c=b;
  return(c);
}
main()
{  double  x=5,y=8,z;
  printf("\nx=%f,y=%f",x,y);
  z=max(x,y);
  printf("\nmax=%f",z);
}
```

程序中第 2～9 行是一个自定义的函数部分，第 2 行定义了一个函数名 max 和指定两个形参 a、b 及其类型，程序第 13 行是一个调用函数语句，其中 max 后面括弧内的 x、y 是实参。x、y 是 main 函数中定义的变量，而变量 a、b 是函数 max 中的形式参数。



调试运行程序

程序运行结果如下：



其中，“x=”、“y=”是程序中原样输出的部分，5.000000 为程序调用之前 x 的值，8.000000 为程序调用之前 y 的值，8.000000 为程序的运行结果。



1．函数的分类

从函数使用的角度来看，C 语言的函数可以分为两类：标准库函数和用户自定义函数。

1）标准库函数与头文件

（1）标准库函数。Turbo C 系统提供了 400 多个标准库函数，按功能可以分为：类型转换函数、字符判别与转换函数、字符串处理函数、标准 I/O 函数、文件管理函数和数学运算函数等。它们的执行效率高，用户需要时，可在程序中直接进行调用。

（2）头文件。C 语言库函数所用到的常量、外部变量、函数类型和参数说明，都在相应的头文件（扩展名为.h）中声明，这些文件通常存放在系统目录“tc\include”中，如：

① stdio.h 文件：标准输入/输出函数所用的常量、结构、宏定义、函数的类型、参数的个数与类型的描述。

② math.h 文件：与数学函数有关的常量、结构及相应的函数类型和参数描述。

③ string.h 文件：与字符串操作函数有关的常量、结构以及相应的函数类型和参数描述。

④ stdlib.h 文件：与存储分配、转换、随机数产生等有关的常量、结构以及相应函数的类型和参数描述。

⑤ ctype.h 文件：字符函数有关的常量、宏定义以及相应函数的类型和参数描述。

使用头文件的方法就是在程序开头使用编译预处理的文件包含#include 语句，具体内容见第六章。

2）用户自定义函数

从函数的形式上看，一个 C 语言程序必须包含一个且只有一个 main 函数，由 main 函数

开始调用其他函数，其他函数也可相互调用，但最终返回主函数结束程序。函数的定义通常分为无参数的函数和有参数的函数两种（特殊的有空函数），如图 5-1 所示。

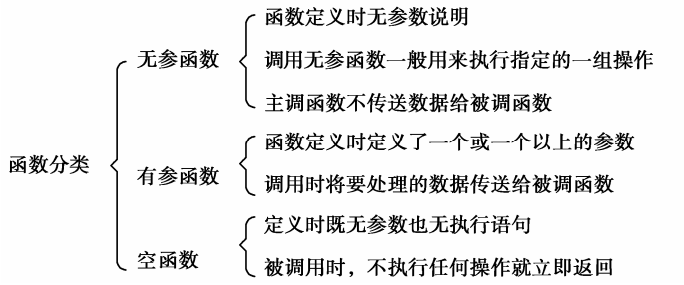


图 5-1　函数的分类及功能



2．函数的定义

函数的定义格式有三种，无参函数的定义、有参函数的定义及空函数的定义。

1）无参函数定义的一般格式

```
类型标志符　函数名()
{　说明部分(可省略)
　　执行部分
　　返回语句(可省略)
}
```

“类型标志符”表示函数值的类型，即函数返回值的类型。无参函数一般不需要返回函数值，因此可以不写类型标志符（或写成空类型 void），下面就是一个无参函数的例子。

```
printstar()                /*定义一个没有参数和返回值的函数 printstar*/
{  printf("* * * * * * * * * * * * *");
}
main()
{  printstar();            /*调用 printstar（）函数*/
}
```

程序运行结果如下：

```
* * * * * * * * * * * * *
```

其中，printstar 是用户定义的函数名，用来输出一排“*”号。

2）有参函数定义的一般格式

```
类型标志符　函数名(形式参数表)
{　说明部分(可省略)
　　执行部分
　　返回语句(可省略)
}
```



有参函数比无参函数多了一个参数表。调用有参函数时，调用函数将赋予这些

参数实际的值。为了与调用函数提供的实际参数区别开，将函数定义中的参数表称为形式参数。

【思考题 5-1】中 max 函数如下所示：

```
double  max(double  a,double  b)      /*函数体的定义*/
{  double  c;                         /*函数体中说明部分*/
   if(a>b)
     c=a;
   else
     c=b;
 return(c);                           /*函数的返回语句*/
}
```

这是一个求 a 和 b 二者中较大者的函数，第一行第一个关键字 double 表示函数值是双精度型的，max 为函数名，括号中有两个形式参数 a 和 b，它们都是双精度型浮点型变量。花括号内是函数体，它包括说明部分和执行部分，在说明部分定义所用的变量。在函数体的语句中求出 c 的值（为 a 与 b 中大者），“return(c);”的作用是将 c 的值作为函数的返回值带回到主调函数中。“return”后面括弧中的值“c”作为函数带回的值（或称函数返回值）。

对于函数有几点说明。

（1）函数名是函数的标识符，遵循 C 语言标识符的命名规则，区分大小写。

（2）函数类型指定所定义函数返回值的类型，可以是简单类型、void 类型或构造类型等。当函数类型为 void 时，表示函数无返回值，相当于其他语言的过程。当函数类型为 int 时，可省略其类型的说明，建议不使用缺省形式类型说明。

（3）形式参数简称形参，处在函数定义部分的函数名后的圆括号中。形式参数表可以为空，表示没有参数（无参函数）；也可以由多个参数组成（本例题中由两个参数 a、b 构成）。当形式参数表中有多个参数时，参数与参数之间用逗号隔开。要特别注意的是，无论函数是否有形式参数，函数名后的圆括号都不可省，并且圆括号之后不能接“;”。

（4）对于有返回值的函数，必须用带表达式的 return 语句来结束函数的运行。C 语言规定，int 类型与 char 类型的函数在定义时可以省略类型符，系统默认这些函数返回值的类型为 int 类型。

（5）函数体是一个复合语句，即用花括号{ }括起来的语句序列。

（6）形式参数属于所在函数的局部变量，其存储类型只能是 auto 型或 register 型，默认为 auto 型。

3）空函数定义的一般格式

```
类型标志符   函数名()
{ }
```

例如，有以下函数 sunny 定义：

```
sunny()
{ }
```

调用此函数时，什么工作也不做，没有任何实际作用。在主调函数中写上“sunny();”表明“这里要调用一个函数”，而现在这个函数没有起作用，等以后扩充函数功能时补充上。在程序设计中往往根据需要确定若干模块，分别由一些函数来实现。而在第一阶段只设计最基本的模块，其他一些次要功能则在以后需要时陆续补上。在编写程序的开始阶段，可以在将来准备扩充功能的地方写上一个空函数，先占一个位置，以后用一个编好的函数代替它。

空函数在程序设计中常常使用。



3．函数的调用

1）函数调用格式

```
函数名(实际参数表)
```

在【思考题 5-1】中，z=max(x,y);为函数的调用语句，max 后面括弧内的 x,y 是实际参数表，我们也把它称为实参。

说明：

- 实际参数（简称实参）的个数多于一个时，各实参之间用逗号隔开。
- 若函数无形参，调用格式为：

```
函数名()
```



（1）实参的个数必须与形参的个数一致。在【思考题 5-1】中，有两个实参分别为 x、y，有两个形参分别为 a、b，实参与形参的格式必须一致。

（2）在定义的函数中，必须指定形参的类型，并且实参的类型必须与形参的类型一一对应。如在【思考题 5-1】中：

```
double  max(double  a,double  b)
{...}
```

形参 a、b 的类型定义为 double，实参为 x、y，定义类型也为 double，且 x 与 a 对应，y 与 b 对应。如果实参 a 为整型而形参 x 为实型，或者相反，则按照不同类型数值的赋值规则进行转换。例如实参 x 的值为 3.5，而形参 a 为整型，则将实数 3.5 转换成整数 3，然后送到形参 a 中。

（3）形参只能是变量，实参可以是常量、变量或表达式。如，max(5)实参为常量，max(3, a+b)实参为表达式，但是要求它们有确定的值，在调用时将实参的值赋给形参。

（4）实参与形参的类型应相同或赋值兼容。【思考题 5-1】中的实参和形参都是 double 类型，这是合法的。

（5）实参可以和形参同名。

可以将【思考题 5-1】中程序的被调用函数 max()改为如下所示：

```
double  max(double  x,double  y)
{  double  c;
   if(x>y)
     c=x;
   else
     c=y;
   return(c);
}
```

形参与实参变量的名称相同，但是不影响程序的结果。



4．函数的调用方式

（1）作为表达式的函数调用。作为表达式的函数调用，本身就是一个表达式，它的值将参与整个表达式的求值过程，因此被调用函数必须是有返回值的。例如：

```
c=max(a,b);
```

（2）作为语句的函数调用。作为语句的函数调用就是在一个语句中单独使用函数调用，也就是在函数调用后加语句结束符“;”，从而构成表达式语句。例如：

```
printf("hello!");
```

（3）函数参数。函数调用作为一个函数的实参。例如，有如下函数调用语句：

```
m=max(a,max(b,c));
```

其中 max(b,c)是一次函数调用，它的值作为 max 另一次调用的实参。m 的值是 a、b 和 c 三者中最大的。

函数调用作为函数的参数，实质上也是函数表达式调用的一种，因为函数的参数本来就要求是表达式形式。



5．函数返回语句return

（1）函数调用过程中，程序执行将从调用处转移到被调用函数处执行，待被调用函数执行完毕，再返回到调用处的下面继续执行。

（2）C 语言的函数根据函数返回值的有无，可分为两类：一类是有返回值的函数，另一类是无返回值的函数。所有函数都可以通过 return 语句，结束被调用函数的执行，返回调用函数。

（3）return 语句的一般格式。

① 有返回值的 return 有两种调用格式：

return　表达式;	或	return (表达式);

有表达式的 return 语句，在实现函数返回时，将表达式的值同时带回调用函数。例如【思考题 5-1】中的语句：

```
return(c);
```

就是将 max 子函数中的结果 c 值返回到主函数中调用语句 c=max(a,b);处。

② 无返回值的 return 调用格式为：

return;

无表达式的 return 语句，只将控制权由被调用函数转回调用函数，而不返回任何值。如果子函数没有任何返回值（即子函数的类型为空类型 void），则使用这样的语句返回。



6．函数返回语句的几点说明

（1）一个 return 语句，只能将一个表达式的值带回调用函数。在一个函数中允许有一个

或多个 return 语句，每个 return 后面的表达式的类型应相同，流程执行到其中一个 return 语句即返回调用函数。

（2）C 语言允许在没有返回值的函数中使用不带表达式的 return 语句，以便立即结束函数的执行，返回到调用函数。当不带表达式的 return 语句位于函数的最末时，允许将 return 语句省去，这时遇到函数体的右花括号也会将流程返回调用函数。

（3）如果 return 语句中表达式值与函数定义的类型不一致，则以函数定义类型为准，并自动将 return 语句中的表达式的值转换为函数返回值的类型。

```
int  max(float  x, float  y)   /*定义有参函数 max*/
{   float  z;
    z=x>y?x:y;
    return(z);
}
```

函数 max 定义为整型，而 return 语句中的 z 为实型，二者不一致，按上述规定，先将 z 转换为整型，然后 max（x,y）带回一个整型值 2 回主调函数 main。如果将 main 函数中的 c 定义为实型，用%f 格式符输出，也是输出 2.000000。

二、函数原型

【思考题 5-2】 分析以下程序，观察函数原型的定义与使用。

```
void  print1(int  n);              /*print1 函数原型*/
main()
{  print1(5);
}
void  print2(int  x,int  y)
{  print1(x);
   print1(y);
}
void  print1(int  n)               /*print1 函数定义*/
{  printf("%d\n",n);
}
```



程序分析

（1）该程序包含三个函数的定义：main 函数、print1 函数和 print2 函数。main 函数和 print2 函数均调用 print1 函数，而 print1 函数的定义位于 main 函数和 print2 函数之后，所以在 main 函数和 print2 函数之前给出 print1 函数的原型。若将 print1 函数的原型改放在 main 函数内，main 函数可以调用 print1 函数，print2 函数将不能调用 print1 函数。所以若将 print1 函数的原型改放在 main 函数内，在 print2 函数内还必须再给出 print1 函数的原型。

（2）若在 main 函数中还要调用 print2 函数，则在调用之前，必须对 print2 函数进行声明，解决这个问题的方法有两个：一是在 main 函数中给出 print2 函数原型，二是在程序的首部、main 函数之前给出 print2 函数原型。



调试运行程序

程序运行结果如下：



小讲堂

1．函数原型的定义

编译程序在处理函数调用时，必须从程序中获得完成函数调用所必需的接口信息。函数原型为函数调用提供接口信息，函数原型是一条程序说明语句。

函数原型的基本格式就是在函数定义格式的基础上去掉函数体，定义格式如下：

```
类型符    函数名(形式参数表);
```

说明：

（1）函数原型必须位于对该函数的调用之前。

（2）函数原型中的函数类型、函数名、参数个数、参数类型和参数顺序必须与函数定义中的相同，但是参数名可以省略。

（3）从格式上可以看出，函数原型所能提供的信息，函数定义也能提供。因此，如果函数定义于函数调用之前，则函数定义本身就同时起到了函数原型的作用。在这种情况下，不必给出函数原型。反之，如果函数定义于函数调用之后，或位于别的源文件中，则必须提前给出函数原型。

小讲堂

2．函数定义和函数原型的比较

1）函数定义和函数原型的作用不同

函数定义是指对函数功能的确立，包括函数名、函数返回值、函数形参及其类型、函数体等，它是完整的、独立的函数单位。函数定义的作用是使被定义的函数“从无到有”；编译程序将根据函数定义生成有关程序代码，使被定义的函数由不存在到存在。

函数原型的作用是为函数调用提供接口信息。函数原型是把函数名、函数类型以及形参的类型、个数和顺序通知编译程序，以便在调用该函数时进行必要的语法或语义检查。

2）函数定义和函数原型在程序中出现的位置不同

函数的定义只能出现在其余函数定义的外部，所有函数的定义都是平行、独立的。

函数原型可以位于所有函数的外部，也可以位于调用函数的函数体的说明部分，但必须位于该函数的调用之前。

3）函数定义和函数原型在程序中出现的次数不同

一个函数在同一个程序中只能定义一次，即函数不能重复定义；但是一个函数的函数原型在同一个程序中可以出现一次，也可以出现多次。

练一练

【练习 5-1】 分析下列程序的输出结果，说明其中的函数定义和函数调用。源程序如下：

```
double  fun(int  n)
{  int  i;
   double  s=1;
   for(i=1;i<=n;i++)
     s*=i;
```

```
    return  s;
}
main()
{  int  i,n;
   double  sum=0;
   printf("\ninput n:");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
     sum+=fun(i);
   printf("\n1!+2!+…+%d!=%.0lf",n,sum);
}
```

解：程序分析如下：

（1）该程序包含两个函数的定义：main 函数和 fun 函数。main 函数是由系统调用执行的，fun 函数是由 main 函数调用的。

（2）fun 函数求自然数 n 的阶乘值并返回。在 main 函数中，首先通过键盘输入一个整数给变量 n，然后调用 fun 函数计算 1!+2!+ …+n！的值。

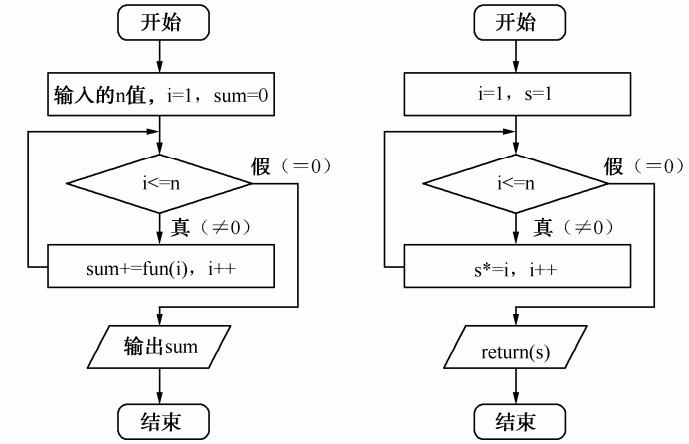
程序流程图如图 5-2 所示。程序运行结果如下：

```
input n:9
1!+2!+...+9!=409113
```

【练习 5-2】 求 x^n

解：数学上经常碰到求 x^n 的问题，即计算 n 个 x 的乘积，当用 C 语言求解方程时会经常用到。

（1）程序设计思路。计算时根据指数运算的定义，用一个函数实现其运算，根据 n 的不同采取不同的计算方法：如果 n＝0，则结果为 1；如果 n<0，则结果为 $1/x^n$；如果 n>0，则结果是 x^n，然后在主函数中调用此函数即可。



（a）主函数的程序流程图　　（b）子函数的程序流程图

图 5-2　【练习 5-1】的程序流程图

（2）算法设计，用自然语言表示的算法如下：

① 接受用户输入的指数和底数；

② 调用函数进行计算；

③ 输出结果。

（3）程序设计与运行，源程序如下：

```
#include "stdio.h"
double pow(double x,int n) /*函数的定义在main()之前时不需要声明*/
{ double y;
  int i;
  y=1;
  if(n==0)
    return(1.0);
  else if(n>0)
  { for(i=1;i<=n;i++)
      y=y*x;
    return(y);
  }
  else
  { for(i=1;i<=n;i++)
      y=y*x;
    return(1/y);
  }
}
main()
{ int x,y;
  printf("input x:");
  scanf("%d",&x);
  printf("input y:");
  scanf("%d",&y);
  printf("result:%f\n",pow(x,y));
}
```

（4）程序运行结果如下：

```
input x:10
input y:5
result:100000.000000
```

本讲小结

这一讲我们学习了函数的定义以及函数的调用格式。在对函数进行定义时，有返回值的函数必须用函数返回语句结束函数的运行。函数可以作为表达式调用，也可以作为语句调用。

想一想

函数在调用过程中可不可以进行嵌套调用和递归调用？如果可以的话应该怎样定义和使用函数呢？

第十二讲　函数的嵌套、递归调用及函数之间的数据传递

学习目标

- 掌握嵌套函数的定义及使用；
- 掌握递归函数的定义及使用；

- 理解参数传递的原理;
- 熟练掌握通过传值方式向函数传递数据的方法;
- 熟练掌握数组参数的使用。

一、函数的嵌套调用

【思考题 5-3】 求 $1^k+2^k+3^k+\cdots+n^k$ 的值。

程序分析

为了解决这个问题，我们设置了三个函数：

（1）power()函数用来求 m^n 的值并返回；

（2）sum_power()函数用来求 $1^k+2^k+3^k+\cdots+n^k$ 的值并返回；

（3）在 main()先由键盘输入变量 k 和 n 的值，然后计算并输出 $1^k+2^k+3^k+\cdots+n^k$ 的值。

程序流程图如图 5-3 所示。

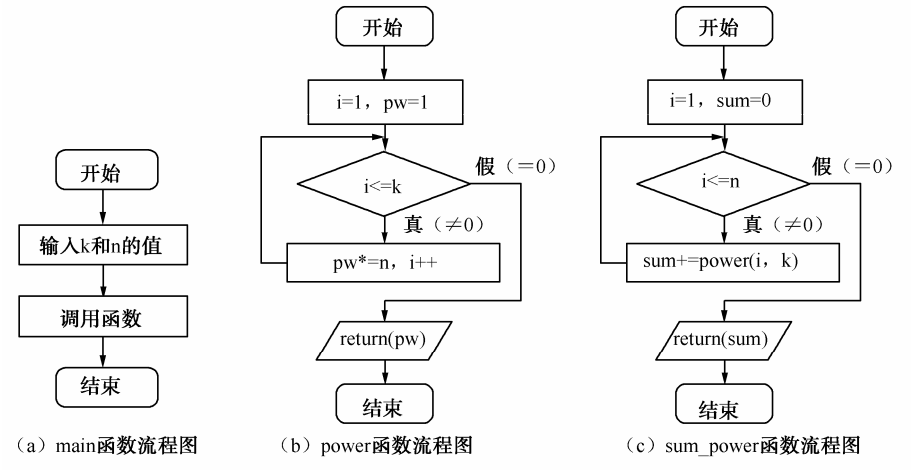


图 5-3 【思考题 5-3】的程序流程图

编写程序代码

```
double  power(int  k, int  n)   /*求 n^k 的值并返回*/
{  int  i;
   double  pw=1;
   for(i=1;i<=k;++i)
     pw*=n;
   return(pw);
}
double  sum_power(int  k, int  n)
/*求 1^k + 2^k + 3^k + …+ n^k 的值并返回*/
{  int  i ;
   double  sum=0;
   for(i=1;i<=n;++i)
     sum+=power(k,i);
   return(sum);
}
main()
```

```
    {   int  k, n;
        printf("\ninput k:");
        scanf("%d",&k);
        printf("\ninput n:");
        scanf("%d",&n);
        printf("\nsum  of  %dth  powers  of  integers  from  1  to  %d=%.0lf\n
",k,n,sum_power(k,n));
    }
```



调试运行程序

程序运行结果如下：

```
input k:3
input n:5
sum of 3th powers of integers from 1 to 5=225
```

3 和 5 为程序输入数据，程序运行结果为 225。



1．函数的嵌套调用

函数的嵌套调用是指在执行被调用函数时，被调用函数又调用了其他函数。

2．函数嵌套调用时需要注意的地方

（1）C 语言程序中的函数定义都是平行、相互独立的。也就是说在一个函数定义的内部，不能定义其他函数，即函数的定义不允许嵌套。

（2）一个函数既可以被其他函数调用，也可以调用其他函数，这就是函数的嵌套调用。



main 函数是由编译程序调用执行的，main 函数可以调用其他函数，但其他函数不能调用 main 函数。

例如，在 main 函数中可以调用 A 函数，在调用 A 函数的过程中可以调用 B 函数；当 B 函数调用结束后返回到 A 函数，当 A 函数调用结束后，再返回到 main 函数，这就是函数的嵌套调用。其调用过程如图 5-4 所示。

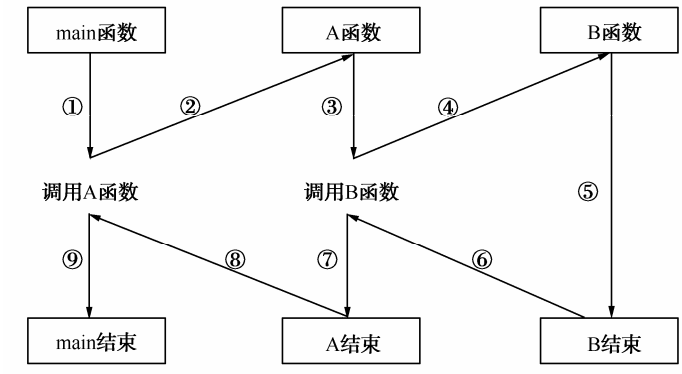


图 5-4　函数的嵌套调用过程示意图

在图 5-4 中，①，②，…，⑨，表示执行嵌套调用过程的序号。即从①开始，先执行 main 函数的函数体中的语句，当遇到调用 A 函数时，由②转去执行 A 函数；③是执行 A 函数的函数体中的语句，当遇到调用 B 函数时，由④转去执行 B 函数，⑤是执行 B 函数的函数体中的所有语句，当 B 函数调用结束后，通过⑥返回到调用 B 函数的 A 函数中；⑦是继续执行 A 函数体中的剩余语句，当 A 函数调用结束后，通过⑧返回到调用 A 函数的 main 函数中；⑨表示继续执行 main 函数的函数体中的剩余语句，结束本程序的执行。

函数嵌套的层数是不受限制的，但嵌套的层数越多，执行效率就越低，因为每次进行函数调用时都要花费时间和占据空间。

二、递归函数及递归调用

【思考题 5-4】 设计递归函数 fact(n)，计算并返回 n 的阶乘值。

程序分析

一个整数 n 的阶乘可表示为：

$$n!=\begin{cases}1 & (n=0 \text{ 或 } n=1)\\ 1*2*\cdots*n & (n>1)\end{cases}$$

阶乘定义还可以表示为：

$$n!=\begin{cases}1 & (n=0 \text{ 或 } n=1)\\ n*(n-1) & (n>1)\end{cases}$$

现在定义一个函数 fact(n)来求 n！时，可以使用如下的方式:

$$fact(n)=\begin{cases}1 & (n=0 \text{ 或 } n=1)\\ n*fact(n-1) & (n>1)\end{cases}$$

从上面我们可以看到，当 n>1 时，fact(n)可以转化为 n*fact(n−1)，而 fact(n−1)与 fact(n)，只是函数参数由 n 变成 n−1；而 fact(n−1)又可以转化为(n−1)*fact(n−2)，……，每次转化时，函数参数减 1，直到函数参数的值为 1 时，1！的值为 1，递归调用结束。

递归调用的过程如图 5-5 所示，假设输入 n=5：

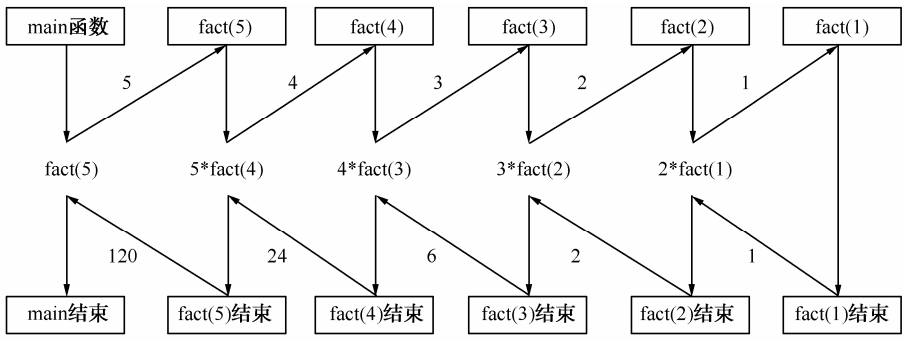


图 5-5 fact 函数的递归调用过程示意图

使用一个 main 函数调用 fact 求 5！，如图 5-5 所示，倾斜箭头表示函数调用，旁边的数字表示传递的参数；下面的箭头表示函数返回，旁边的数字表示函数返回值。fact 函数反复调用自身；fact(5)调用 fact(4)，fact(4)调用 fact(3)，fact(3)调用 fact(2)，……，参数逐次减小，

当最后调用 fact(1)时，结束调用，于是开始逐级完成乘法运算，最后计算出 5!结果为 120。

编写程序代码

```
double  fact(int  n)
{  if( n==0 || n==1)
     return(1);
   else
     return(n*fact(n-1));
}
main()
{  int  num;
   printf("\ninput num:");
   scanf("%d",&num);
   printf("\nfact(%d)=%.0lf",num,fact(num));
}
```

调试运行程序

程序运行结果如下：

```
input num : 5
fact(5)=120
```

其中：5↙表示输入的阶乘数，120 为 5!的运行结果。

小讲堂

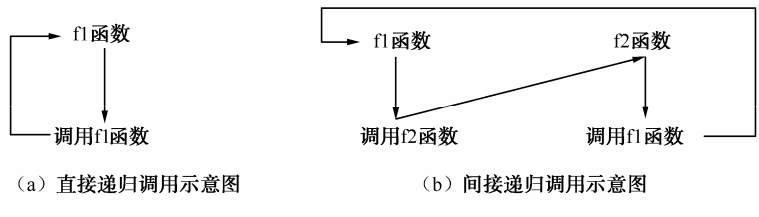
1．递归函数

（1）递归函数的概念。函数通过其函数体中的语句直接或间接地调用自身，称为递归调用，这样的函数称为递归函数。

（2）直接递归与间接递归。

① 如果函数 f1 直接调用它本身，称为直接递归调用，如图 5-6（a）所示。

② 如果函数 f1 调用函数 f2，函数 f2 调用函数 f1，则称为间接递归调用，如图 5-6（b）所示。



（a）直接递归调用示意图　　（b）间接递归调用示意图

图 5-6　递归调用示意图

从图 5-6 可以看到，这两种递归调用都是无终止的自身调用。显然，程序中不应出现这种无终止的递归调用，而只应出现有限次数的、有终止的递归调用，我们来看看递归调用必须满足的条件。

小讲堂

2．递归算法必须满足的两个条件

无论是直接还是间接递归，两者都是无终止的调用自身。要避免这种情况的发生，使用递归解决的问题应满足两个基本条件。

（1）问题的转化。有些问题不能直接求解或难以求解，但它可以转化为一个新问题，这个新问题相对较原问题简单或更接近解决方法。这个新问题的解决与原问题一样，可以转化为下一个新问题，……

（2）转化的终止条件。原问题到新问题的转化是有条件的，次数是有限的，不能无限次数地转化下去。这个终止条件也称为边界条件，相当于递推关系中的初始条件。

小讲堂

3．递归与递推的优点与缺点

采用递归方法在解决问题的时候有很多优点，而递推方法也有自己的特点，现比较如下：

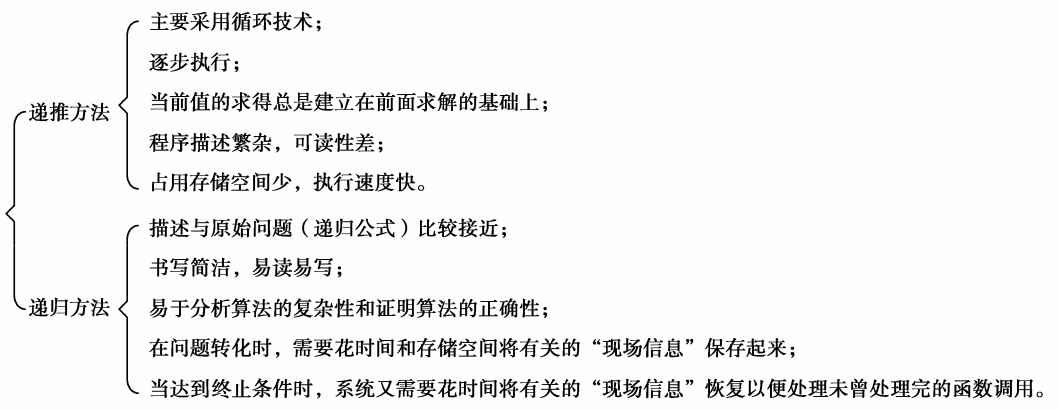


图 5-7　递归与递推的优缺点比较

三、实参-形参之间的数据传递（值传递方式）

【思考题 5-5】　阅读下面程序，注意观察程序的运行结果。

```
void  swap(int  x, int  y);
main()
{  int  a=5,b=8;
   printf("\n(1)a=%d,b=%d\n",a,b);
   swap(a,b);
   printf("\n(4)a=%d,b=%d\n\n",a,b);
}
void  swap(int  x,int  y)
{  int  t;
   printf("\n(2)x=%d,y=%d\n",x,y);
   t=x;
   x=y;
   y=t;
   printf("\n(3)x=%d,y=%d\n",x,y);
```

```
}
```

程序分析

main 函数调用 swap 函数时，将变量 a 和变量 b 的值传递给对应形参 x 和形参 y。在 swap 函数中，交换了形参 x 和形参 y 的值。但由于在 C 语言中，参数中的数据只能由实参单向传递给形参，形参数据的变化并不影响对应实参的值，因此该程序不能通过 swap 函数将 main 函数中的变量 a 和 b 的值进行交换。

调试运行程序

程序运行结果如下：

```
(1)a=5, b=8
(2)x=5, y=8
(3)x=8, y=5
(4)a=5, b=8
```

小讲堂

1．对函数的形参和实参的几点说明

（1）在调用函数时，函数的实参和形参在个数、类型和次序上要一一对应。

（2）对于非 void 类型的函数，函数的调用只能通过 return 语句得到一个返回值；对于 void 类型的函数，调用函数不能返回值。

（3）形参在函数未被调用时，系统并不给它们分配存储单元，只有在发生函数调用时形参才被分配临时存储单元，但在调用结束后，系统自动释放形参占用的存储单元。

（4）实参可以是常量、变量或表达式，但必须有确定的值，以便传递给形参。

（5）形参一定是变量，当然可以是指针类型，如数组名（它代表数组的首地址）和指针变量（见第七章）。

（6）值传递。如果实参是一个非指针类型的变量（包括数组名），此时形参和实参的数值传递是“单向的值传递”，即实参仅把它的值传递给对应的形参，形参的值即使在函数中发生了改变，形参的值也不能传递给实参。

四、实参-形参之间的数据传递（数组作函数参数）

【思考题 5-6】 已知某个学生 5 门课程的成绩，求其平均成绩。观察数组名作函数参数的使用。

编写程序代码

```
float  aver(float  a[5])                /*求平均值函数*/
{  int  i;
   float  av,s=0;
   for(i=0;i<5;i++)
     s+=a[i];                           /*循环累加各数组元素的值*/
   av=s/5;                              /*求平均值*/
   return  av;
}
main()
{  float  sco[5],av;
```

```
    int  i;
    printf("\ninput 5 scores:\n");
    for(i=0;i<5;i++)
      scanf("%f",&sco[i]);              /*输入 5 门课程的成绩*/
    av=aver(sco);                       /*调用函数，实参为数组名*/
    printf("average score is %5.2f\n",av);
}
```



程序分析

（1）用数组名作函数参数，应该在主调函数和被调用函数中分别定义数组，【思考题 5-6】中 sco 是实参数组名，a 是形参数组名，分别在其所在函数中定义，不能只在一处定义。

（2）实参数组与形参数组类型应一致。

（3）在被调用函数中声明了形参数组的大小为 5，但在实际上，指定其大小是不起任何作用的，因为 C 编译对形参数组大小不做检查，只是将实参数组的首地址传给形参数组。因此，sco[n]和 a[n]指的是同一单元。

（4）形参数组也可以不指定大小，在定义数组时在数组名后面跟一个空的方括弧，即 float aver(float a[])。有时为了在被调用函数中处理数组元素的需要，可以另设一个参数，传递需要处理的数组元素的个数。

（5）用数组名作函数实参时，不是把数组元素的值传递给形参，而是把实参数组的起始地址传递给形参数组，这样两个数组就共占同一段内存单元。



调试运行程序

程序运行结果如下：

```
input 5 scores:
65 98 56 78 82
average score is 75.80
```

其中，65、98、56、78、82 为输入的成绩，最终输出成绩平均值，并保留 2 位有效小数。



2．数组元素作函数参数

（1）数组元素就是下标变量，它与普通变量并无区别。数组元素只能用做函数实参，其用法与普通变量完全相同：在发生函数调用时，把数组元素的值传送给形参，实现单向值传送。

（2）用数组元素做实参时，只要数组类型和函数的形参类型一致即可，并不要求函数的形参也是下标变量。换句话说，对数组元素的处理是按普通变量对待的。

（3）在普通变量或下标变量做函数参数时，形参变量和实参变量是由编译系统分配的两个不同的内存单元。在函数调用时发生的值传送是把实参变量的值赋予形参变量。



3．数组名作函数参数的几点说明

（1）数组名作函数参数时，要求形参和相对应的实参都必须是类型相同的数组（或指向数组的指针变量），都必须有明确的数组说明。

（2）用数组名作函数参数，应该在调用函数和被调用函数中分别定义数组，且数据类型

必须一致，否则结果将出错。例如，形参数组为 float　a[]，实参数组为 float　sco[]，它们的数据类型相同。

（3）C 编译系统对形参数组大小不做检查，所以形参数组可以指定大小或不指定大小。例如，形参数组 a[5]可写成 a[]。



如果指定形参数组的大小，则实参数组的大小必须大于等于形参数组，否则会因形参数组的部分元素没有确定值而导致计算结果错误。



4．用多维数组名作函数参数

多维数组元素可以作为实参，这点与前述相同。可以用多维数组名作为实参和形参，在被调用函数中对形参数组定义时可以指定每一维的大小，也可以省略第一维的大小说明。如 int　arr[3][5];或 int　arr[][5];二者都合法而且等价。但是不能把第二维以及其他高维的大小说明省略，如 int　arr[3][];或 int　arr[][]; 均不合法。

练一练

【练习 5-3】　设计递归函数 gcd(x,y)，求 x 和 y 的最大公约数。

解：程序分析：如果 x%y==0，x 和 y 的最大公约数就是 y；否则求 x 和 y 的最大公约数等价于求 y 与 x%y 的最大公约数。这时可以把 y 当做新的 x，x%y 当做新的 y，问题又变成了求新的 x 与 y 的最大公约数。它又等价于求新的 y 与 x%y 的最大公约数，……，如此继续，直到新的 x%y==0 时，其最大公约数就是新的 y。

例如，求 48 与 36 的最大公约数，等价于求 36 与 48%36 的最大公约数，即求 36 与 12 的最大公约数。此时 36%12==0，最大公约数就是 12。

求 x 和 y 的最大公约数，用函数 gcd(x,y)表示如下：

$$\text{gcd}(x,y)=\begin{cases} y & (x\%y==0) \\ \text{gcd}(y,x\%y) & (x\%y!=0) \end{cases}$$

程序源代码如下：

```
int  gcd(int  x,int  y)
{  if(x%y==0)
      return(y);
   else
      return(gcd(y,x%y));
}
main()
{  int  x,y;
   printf("\ninput x:");
   scanf("%d",&x);
   printf("\ninput y:");
   scanf("%d",&y);
   printf("\ngcd(%d,%d)=%d",x,y,gcd(x,y));
}
```

程序运行结果如下：

```
input x : 48
input y : 36
gcd(48,36)=12_
```

其中，“input x”和“input y”是提示信息，分别为 x、y 输入 48 和 36 两个数，12 为两数的最大公约数，即程序的运行结果。

【练习 5-4】 任意输入 3 个整数，利用函数的嵌套调用求出 3 个数中的最小值。

解：程序设计思路如下：在 main()函数中接受用户输入的 3 个数，再通过函数调用来求出最小值。在 main()外先定义函数 int mintwo(int,int) 求出两个整数中的最小值，然后定义 int minthree(int,int,int)函数，并在此函数中调用 mintwo(int,int)函数，求出 3 个数中最小值。

程序源代码如下：

```
#include <stdio.h>
main()
{ int x,y,z,min;
  int mintwo(int,int),minthree(int,int,int);
  printf("\nplease input three integers:");
  scanf("%d%d%d",&x,&y,&z);
  min=minthree(x,y,z);                    /*函数调用语句，3 个实参进行传递*/
  printf("min is %d",min);
}
int mintwo(int a,int b)                   /*函数定义*/
{ return(a<b?a:b);                        /*使用条件语句，并将计算结果返回*/
}
int minthree(int a,int b,int c)
{ int z;
  z=mintwo(a,b);                          /*函数内嵌套调用函数 mintwo()*/
  z=mintwo(z,c);
  return(z);
}
```

程序运行结果如下：

```
please input three integers:12 56 -9
min is -9_
```

【练习 5-5】 有一个 3×4 的矩阵，求所有元素中的最大值。

解：

```
fmax(int arr[ ][4])                       /*定义 fmax 函数，二维数组 arr 作形参*/
{ int i,j,k,max ;
  max=arr[0][0];
  for( i=0;i<3;i++)
    for(j=0;j<4;j++)
      if(arr[i][j]>max)
        max=arr[i][j];
  return(max);
}
main()
{ int a[3][4]={{1,3,5,7},{2,4,6,8},{15,17,34,12}}; /*定义二维数组 a*/
  printf("max value is %d\n",fmax(a));   /*调用 fmax 函数，二维数组 a 作实参*/
}
```

程序运行结果如下：



【练习 5-6】 设计函数 even，验证任意偶数为两个素数之和并输出这两个素数。

解：程序的算法分析如下：

（1）在 main 函数中，首先从键盘输入一个不小于 4 的偶数 n，然后调用函数 even 将 n 拆分两个素数的和，并输出这两个素数。

（2）函数 prime 的功能是判断参数 n 是否为素数。如果参数 n 为素数，返回 1；否则，返回 0。

（3）函数 even 的功能是将参数 n 拆分为两个素数的和，并输出两个素数。该函数的算法描述如下：

① i 初值为 2。

② 判断 i 是否是素数。若是，执行步骤③；若不是，执行步骤⑤。

③ 判断 n−i 是否是素数。若是，执行步骤④；若不是，执行步骤⑤。

④ 输出结果，返回调用函数。

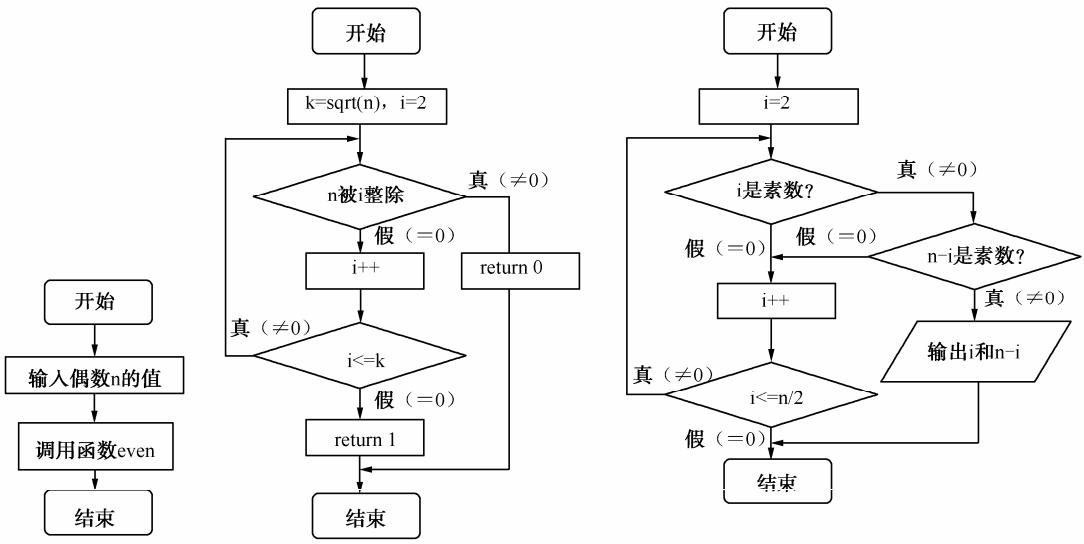
⑤ 使 i 增 1。

⑥ 重复执行步骤②。

主函数及两个子函数的流程图如图 5-8 所示。

源程序如下：

```
#include "math.h"
int prime(int n);
void even(int n );
main()
{ int n;
  printf("\ninput n:");
  do
  { scanf("%d",&n);
  }while(n>=4&&n%2!=0);
  even(n);
}
void even (int n)
{ int i=2;
  do
  { if(prime(i)&&prime(n-i))
    { printf("\n%d=%d+%d\n",n,i,n-i );
      return;              /*结束 even 函数的执行，返回调用函数*/
    }
  i++;
  }while( i<=n/2 );
}
int prime(int n)
{ int i=2,k;
  k=sqrt(n);
  do
  { if(n%i ==0)
    return  0;             /*结束 prime 函数的执行，返回调用函数*/
    i++;
  }while(i<=k);
  return 1;                /*结束 prime 函数的执行，返回调用函数*/
}
```



（a）main函数流程图　　（b）even函数流程图　　（c）prime函数流程图

图 5-8　【练习 5-6】的程序流程图

程序运行结果如下：

```
input n : 16
16=3+13
```

本讲小结

这一讲我们学习了函数的嵌套、函数的递归调用、函数的值传递方式和以数组作为函数参数。函数的递归调用为直接或间接调用自己。函数调用时通常以传值的方式传递参数，改动形参变量的值不会影响到对应的实参变量。数组作为函数参数有两种方式：一种是把数组元素（又称下标变量）作为实参使用；另一种是把数组名作为函数的形参使用。

想一想

什么情况下我们采用函数的嵌套调用方式？什么情况下采用函数的递归调用方式？子函数写在主函数的前面还是后面时需要用到函数的原型？变量定义的位置可以不同，那么变量的作用域以及存储类型如何使用呢？

第十三讲　变量作用域及存储类型、内部函数和外部函数

学习目标

- 掌握局部变量和全局变量的作用域和存储类型；
- 掌握内部函数和外部函数的定义及使用；
- 掌握如何运行一个多文件程序。

在 C 语言中，标识符必须先定义后使用，但定义语句应该放在什么位置？在程序中，一

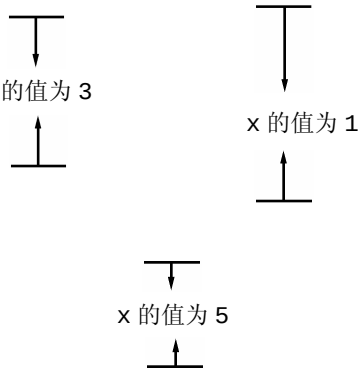
个定义了的标识符是否随处可用？这些问题涉及标识符的作用域。经过赋值的变量是否在程序运行期间总能保存其值？这涉及变量的生存期。在本讲中，特别注意“定义”和“说明”这两个词。“定义”是指给变量分配确定的存储单元，“说明”只是说明变量的性质，而不分配存储空间。

一、作用域和生存期

【思考题 5-7】 分析下列程序的输出结果，注意其中的自动变量。

编写程序代码

```
main()
 {  int  x=1;
    {  int  x=3;
       void  prt(void);
       prt();
       printf("(2)  x=%d\n",x);
    }
    printf("(3)  x=%d\n",x);
}
 void  prt(void)
{  int  x=5;
   printf("(1)  x=%d\n", x);
}
```



程序分析

（1）该程序中定义了 3 个局部变量，名字均为 x。因为它们的作用域互不相同，所以 3 个同名变量互不干扰。

（2）初值为 1 的变量 x 和初值为 3 的变量 x 的作用域嵌套，在初值为 3 的变量 x 的作用域内，初值为 1 的变量 x 无效。

（3）初值为 5 的变量 x，只在 prt 函数中有效，而在 main 函数中无效。

调试运行程序

程序运行结果如下：

```
(1)  x=5
(2)  x=3
(3)  x=1
```

小讲堂

在第二章中我们学过，变量定义的一般格式为：

```
存储类型标识符  类型标识符  变量名;
```

其中类型标识符说明变量的数据类型，如整型、实型等。而 C 语言的数据有四种存储类型，分别由四个关键字（称为存储类型标识符）表示：auto（自动）、extern（外部）、static（静态）和 register（寄存器）。例如，在前面的程序中所使用的变量，它们的存储类型均为 auto 类型。

1．变量的作用域

在 C 语言中，由用户名命名的标识符都有一个有效的作用域，所谓标识符的“作用域”是指程序中的某一部分，在这一部分中，该标识符是有定义的，可以被 C 语言编译程序和连接程序所识别。

C 程序中每个变量都有自己作用域。例如，在一个函数内定义的变量，不能在其他函数中使用。变量的作用域与其定义语句在程序中出现的位置有着直接的关系。注意，在同一个作用域内，不允许有同名的变量出现，而在不同的作用域内，允许有同名的变量出现。

依据变量作用域的不同，C 语言变量可以分为局部变量和全局变量两大类。在函数内部或复合语句内部定义的变量，称为局部变量。函数的形参也属于局部变量。在函数外部定义的变量，称为全局变量。有时将局部变量称为内部变量，全局变量称为外部变量。



2．变量的生存期

C 程序通常存放在内存和寄存器中，少数的寄存器只能用来存放一些被反复加工的数据，所以内存是 C 程序的主要存储区。

C 程序占用的内存空间通常分为三部分：**程序区**、**静态存储区**和**动态存储区**。其中程序区中存放的是可执行程序的机器指令；静态存储区中存放的是需要占用固定存储单元的数据；动态存储区中存放的是不需要占用固定存储单元的数据。C 语言允许程序员在静态存储区、动态存储区、寄存器中开辟变量的存储空间。

所谓变量的生存期是指变量值在程序运行过程中的存在时间。C 语言变量的生存期可以分为静态生存期和动态生存期。存放在静态存储区的变量具有静态生存期；存放在动态存储区的变量具有动态生存期，即其生存期从变量被定义开始，到所在复合语句运行结束时为止。

二、局部变量的作用域和存储类型

【思考题 5-8】 分析下列程序的输出结果，注意其中的静态局部变量和自动变量中值的变化。

```
void  test();
main()
{  test();
   test();
   test();
}
void  test()
{  int  x=0;
   static  int  y=0;
   x++;
   y++;
   printf("x=%d,y=%d\n",x,y);
}
```



（1）x 是自动变量，每调用一次 test 函数，x 都被动态分配存储空间（调用开始时分配空

间，调用结束后回收空间），赋初值均为 0，这样，函数 test 连续调用 3 次，输出变量 x 的值均是 x=1。

（2）y 是静态局部变量，第一次调用 test 函数时，系统就为变量 y 分配存储空间，并赋初值为 0，调用结束后，变量 y 的存储空间不被回收。第二次和第三次调用 test 函数时，变量 y 的值就可以被传递或继承。这样，函数 test 连续调用了 3 次输出的变量 y 的值分别是 y=1，y=2，y=3。


调试运行程序

程序运行结果如下：

```
x=1, y=1
x=1, y=2
x=1, y=3
```



3．局部变量的定义

定义于函数内部或复合语句内部的变量均称为局部变量。其作用域为复合语句作用域，也就是说只有在本函数内才能使用它们，在此函数以外是不能使用这些变量的。

局部变量定义时，可以使用 auto、static 和 register 三种存储类型符。

复合语句作用域的范围是从变量定义处开始，到复合语句的结束处为止。

```
int  f1(int  a)                  /*函数 f1*/
{  int  b,c;
   ⋮                             /*a、b、c 作用域：仅限于函数 f1()中*/
}
char  f2(int x)                  /*函数 f2*/
{  int  y,z;                     /*x、y、z 作用域：仅限于函数 f2()中*/
}
main()                           /*主函数 main*/
{  int  m,n;
   ⋮                             /*m、n 作用域：仅限于函数 main()中*/
}
```

说明：

（1）主函数 main 中定义的变量 m、n 也只在主函数中有效，而不因为是在主函数中定义的就在整个文件或程序中有效。主函数也不能使用其他函数中定义的变量。

（2）不同函数中可以使用相同名字的变量，它们代表不同的对象，互不干扰。例如，在 f1 函数中定义了变量 b 和 c，倘若在 f2 中也定义变量 b 和 c，它们在内存中占不同的单元，互不混淆。

（3）形式参数也是局部变量，其他函数不能调用。

（4）在一个函数内部，可以在复合语句中定义变量，这些变量只在复合语句中有效，这些复合语句也可称为“分程序”或“程序块”。

注意 函数体（或复合语句）中局部变量的定义语句必须放在全部可执行语句之前。

使用局部变量需要注意的问题如下：

（1）在一个函数内部不能使用其他函数内部定义的局部变量。

（2）在不同函数内（或不同的复合语句内）可以定义同名的局部变量，它们代表不同的对象，互不干扰。

（3）在嵌套的复合语句内，如果内层与外层有同名的局部变量，则在内层范围内只有内层的局部变量有效，外层的局部变量在此范围内无效。



4．自动变量（auto）

自动变量：用 auto 定义的变量称为自动变量，常常缺省 auto。如果局部变量未用任何关键字进行存储类型说明，默认为自动变量。定义格式如下：

```
[auto]  类型标识符  变量表;
```

自动变量具有动态生存期，即其生存期从变量被定义开始，到所在复合语句运行结束时为止。

（1）在函数中定义的自动变量只在该函数内有效，函数被调用时分配存储空间，调用结束就释放。在复合语句中定义的自动变量只在该复合语句中有效，退出复合语句后，便不能再使用，否则将引起错误。

（2）定义但并不初始化，则其值是不确定的。如果初始化，则赋初值操作是在调用时进行的，且每次调用都要重新赋一次初值。

（3）由于自动变量的作用域和生存期都局限于定义它的个体内（函数或复合语句），因此不同的个体中允许使用同名的变量而不会混淆。即使在函数内定义的自动变量，也可与该函数内部的复合语句中定义的自动变量同名。



系统不会混淆，并不意味着人也不会混淆，所以应尽量少用同名自动变量！



5．寄存器变量（register）

用 register 定义的变量是一种特殊的自动变量，称为寄存器变量。这种变量建议编译程序将变量中的数据存放在寄存器中，而不像一般的自动变量那样，占用内存单元，可以大大提高变量的存取速度。

一般情况下，变量的值都是存储在内存中的。为提高执行效率，C 语言允许将局部变量的值存放到寄存器中，这种变量就称为寄存器变量。

定义格式如下：

```
register    类型标识符    变量表;
```

（1）只有局部变量才能定义成寄存器变量，全局变量不能。

（2）对寄存器变量的实际处理随系统而异。例如，计算机上的 MSC 和 TC 将寄存器变量实际当做自动变量处理。

（3）允许使用的寄存器数目是有限的，不能定义任意多个寄存器变量。



使用 register 定义局部变量，只是一种请求或建议，当没有足够的寄存器来存放

指定的变量，或编译程序认为指定的变量不适合放在寄存器中时，将自动按 auto 变量来处理。



6．静态局部变量（static）

在函数体（或复合语句）内部，用 static 来定义一个变量时，可以称该变量为静态局部变量。定义格式如下：

```
static  类型标识符  内部变量表;
```

（1）静态局部变量属于静态存储。在程序执行过程中，即使所在函数调用结束也不释放。换句话说，在程序执行期间，静态局部变量始终存在，但其他函数是不能引用它们的。

（2）定义但并不初始化，则自动赋以“0”（整型和实型）或‘\0’（字符型），且每次调用它们所在的函数时，不再重新赋初值，只是保留上次调用结束时的值。

（3）何时使用静态局部变量：

① 需要保留函数上一次调用结束时的值；

② 变量只被引用而不改变其值。

三、全局变量的作用域、存储类型及多文件程序的运行

【思考题 5-9】 分析下列程序的输出结果，注意其中的自动变量、静态全局变量和全局变量，并注意多文件程序的编辑及运行方法。

项目文件 ff.prj 的内容如下：

```
f1.c
f2.c
f3.c
f4.c
```

文件 f1.c 中的内容如下所示：

```
int  a;                                  /*全局变量 a 的定义*/
void  fun1(),fun2(),fun3();
main()
{  a=35;
   fun1();
   printf("main():a=%d\n",a);
   fun2();
   printf("main():a=%d\n",a);
   fun3();
   printf("main():a=%d\n",a);
}
```

文件 f2.c 中的内容如下所示：

```
static  int  a;                          /*静态全局变量 a 的定义*/
void  fun1()
{  a=27;
   printf("fun1():a(static)=%d\n",a);
}
```

文件 f3.c 中的内容如下所示：

```
void  fun2()
```

```
{  int  a=16;
   printf("fun2():a(auto)=%d\n",a);
}
```

文件 f4.c 中的内容如下所示：

```
extern  int  a;                          /*全局变量 a 的说明*/
void  fun3()
{  a=48;
   printf("fun3():a(extern)=%d\n",a);
}
```



程序分析

（1）源文件 f1.c 中定义的全局变量 a，由于没有使用 static 存储类型符，可以在源文件 f2.c,f3.c,f4.c 中使用。该程序的源文件 f4.c 中要使用全局变量 a，所以应在 f4.c 文件的首部，用 extern 关键字对全局变量 a 进行说明。

（2）源文件 f2.c 中定义的全局变量 a，使用了 static 存储类型符，即为静态全局变量，具有文件作用域，所以程序中的其余源文件不能使用它。

（3）在源文件 f3.c 中，函数 fun2 定义的变量 a 是局部变量，其作用域是从定义处开始，到所在的复合语句结束，即程序中的所有其他函数都不能使用它。



调试运行程序

程序运行结果如下：

```
fun1():a(static)=27
main():a=35
fun2():a(auto)=16
main():a=35
fun3():a(extern)=48
main():a=48
```



前面已提到一个程序往往由多个文件组成，那么如何把这些文件编译连接成一个统一的可执行文件并运行呢？

7．使用turbo C集成环境运行多文件程序

【思考题 5-9】的程序，应该如何进行呢？它包含 4 个文件。

（1）先后输入并编辑 4 个文件，并分别以文件名 file1.c、file2.c、file3.c、file4.c 存储在磁盘上。

（2）在编译状态下，建立一个“项目文件”，它不包括任何程序语句，而只包括组成程序的所有文件名，即：

```
file1.c
file2.c
file3.c
file4.c
```

（说明：扩展名.c 可以省略，4 个文件顺序任意，也可以连续写在同一行上。）



注意　如果这些源文件不在当前目录下，应指出路径。

（3）将以上内容存盘，文件名自定，但扩展名必须为.prj（表示为 project 文件）。【思

考题 5-9】中项目文件名为 ff.prj。在 Turbo C 主菜单中选择 Project 菜单，按回车键后出现下拉菜单，找到其中的 Project Name 项并按回车键，屏幕上会出现一个对话框，询问项目文件名，如图 5-9 所示。



图 5-9 “Project Name”对话框

输入项目文件名 ff.prj 以代替*.prj，此时子菜单中的 Project Name 后面会显示出项目文件名 ff.prj，表示当前准备编译的是 ff.prj 中包括的文件。

（4）按 F9 功能键，进行编译连接，系统先后将 4 个文件翻译成目标文件，并把它们连接为一个可执行文件 ff.exe（文件名主干与项目文件相同）。

（5）按 Ctrl+F9 快捷键，即可运行可执行文件 ff.exe。

小讲堂

8．全局变量说明

全局变量是在函数外部任意位置上定义的变量，它的作用域是从变量定义的位置开始，到整个源文件结束。

全局变量通常用于记录程序中的全局信息，是函数之间（尤其是没有调用、被调用关系的函数之间）交换数据信息的媒介。当一个全局变量定义于文件首部时，该文件中的所有函数对全局变量的操作都是有效的。如果前面的函数改变了全局变量的值，那么后面的函数引用该变量时，得到的就是被改变后的值。因此，在全局变量的使用中要特别小心，以免造成意料不到的改变。

全局变量具有静态生存期，即变量值存在于程序的整个运行期间，因此是一种静态变量。一切静态变量，如果在定义它的时候未进行初始化，则自动地被初始化为 0。

小讲堂

9．不用static存储类型符定义的全局变量

如果定义全局变量时没有使用 static，则该变量具有程序作用域，即该变量不但可以被其所在的源文件中的函数访问，也可以被同一程序的其他源文件中的函数访问。在后一种情况下，如果程序的某一个源文件中的函数要访问该变量，必须在访问该变量的源文件中提前用 extern 对该变量进行说明，称为外部说明。外部说明的作用是声明该变量已在其他源文件中定义，通知编译程序不必再为它开辟存储单元。例如在【思考题 5-9】中的文件 f1.c 中的全局变量定义语句：int a；定义了一个全局变量 a，可以在该程序的各位置使用。

小讲堂

10．全局变量的说明语句

如果全局变量和访问它的函数是在同一个文件中定义的，且全局变量定义于要访问它的函数之后，也必须在访问该变量之前用 extern 对该全局变量进行说明，这时其作用域从 extern 说明处起，延伸到该函数末尾。所以为了处理的方便，全局变量通常定义于文件首部，位于

所有函数定义之前。

如果在其他文件中要使用该全局变量，则需要在使用的文件首部说明一下该变量，说明语句的格式为：

```
extern  类型标识符  内部变量表;
```

例如在【思考题 5-9】中的文件 f1.c 中定义的全局变量 a，在其他文件中要使用变量 a，需要在使用的文件首部说明它为外部变量。在文件 f4.c 中首部的全局变量说明语句：extern int a;，就说明 a 为一个外部全局变量（注意是说明，不是定义），变量 a 是在其他文件中被定义的。



注意 外部全局变量说明语句的功能是说明一个在其他文件中已定义的一个全局变量，而不是重新定义一个全局变量。



11. 使用static定义的静态全局变量

如果定义全局变量时使用 static 存储类型符，此变量可称做静态全局变量。静态全局变量具有文件作用域，只限于本源文件使用，不能被其他源文件使用，即只允许该变量所在的源文件中的函数访问，而不允许其他源文件中的函数访问。所以【思考题 5-9】中文件 f2.c 中的语句：static int a；就定义了一个静态全局变量 a，所以该变量只能在本文件内部使用。

四、内部函数与外部函数

【思考题 5-10】 分析下列程序的输出结果，注意其中的外部函数说明语句。

（1）文件 mainf.c 中代码为：

```
main()
{  extern  input(char  str[50]);
   /*声明在本函数中将要调用的在其他文件中定义的 3 个函数*/
   extern  del(char  str[],char  ch);
   extern  output(char  str[50]);
   char  c;
   char  str[50];
   input(str);
   scanf("%c",&c);
   del(str,c);
   output(str);
}
```

（2）文件 subf1.c 中代码为：

```
#include  <stdio.h>                    /*定义外部函数*/
input(char  str[50])
{  gets(str);}
```

（3）文件 subf2.c 中代码为：

```
#include  <stdio.h>                    /*定义外部函数*/
del(char  str[],char  ch)
{  int  i,j;
   for(i=j=0;str[i]!='\0';i++)
```

```
        if(str[i]!=ch)
        str[j++]=str[i];
    str[j]= '\0';
}
```

（4）文件 subf3.c 中代码为：

```
#include  <stdio.h>                    /*定义外部函数*/
output(char  str[ ])
{  printf("%s",str);}
```



程序分析

整个程序由 4 个文件组成，每个文件包含一个函数。

主函数是主控函数，除声明部分外，由 4 个函数调用语句组成。其中 scanf 是库函数，另外 3 个是用户自己定义的函数。函数 del 的作用是根据给定的字符串 str 和准备删除的字符 ch，对 str 字符串作删除处理。

算法如下：对 str 数组的字符逐个检查，如果不是被删除的字符就将它存放在数组中。从 str[0]开始逐个检查数组元素值是否等于指定要删除的字符，若不是就留在数组中，若是就不保留。应该使 str[0]赋给 str[0]，str[1]赋给 str[1]，str[2]赋给 str[2]，str[3]赋给 str[3]，str[5]赋给 str[4]，……请注意分析如何控制 i 和 j 的变化，以便使被删除的字符不保留在原数组中。我们只用一个数组，只把不被删除的字符保留下来，由于 i 总是大于或等于 j，因此最后保留下来的字符不会覆盖未被检测处理的字符，最后将字符'\0'也复制到被保留的字符后面。

程序中 3 个函数都定义为外部函数。在 main 函数中用 extern 声明在 main 函数中用到的 input、del、output 是在其他文件中定义的外部函数。



调试运行程序

程序运行结果如下：



其中：

abcdef　（abcdef 为输入的字符串）
d　（d 为要删除的字符）
abcefg　（输出已删去指定字符的字符串的结果）



12．内部函数

定义一个函数时，若在函数类型前加上存储类型符 static 时，则称此函数为内部函数或静态函数。内部函数定义一般格式：

```
static  类型标识符  函数名(函数参数表)
{…}
```

内部函数只限于本文件的其他函数调用它，而不允许其他文件中的函数对它进行调用，所以内部函数的作用域是文件级的。在同一程序的不同源文件中，可以有同名的内部函数出现，它们代表不同的函数，互不干扰。所以使用内部函数，可以避免不同编译单位因函数同

名而引起的混乱。若强行调用静态函数，将会产生出错信息。例如：

```
static int max(int x,int y)
{return(x>=y?x:y;);}
```

则 max 函数只允许被同一源文件中的函数调用。在多人分工合作下，这种对函数作用域的限制很有用。同一系统中的由不同人员设计的文件中即使存在着同名函数，也互不干扰，因此设计人员可以放心地为函数取自己喜欢的名字。



13. 外部函数

函数在本质上都具有外部性质，除了内部函数（静态函数）之外，其余的函数都可以被同一程序的其他源文件调用。当定义一个函数时，若在函数返回值的类型前加上存储类型符 extern 时，称此函数为外部函数。关键字 extern 可以省略。

外部函数定义的一般格式如下：

```
[extern]  类型标识符  函数名(函数参数表)
{...}
```

使用 extern 声明就能够在一个文件中调用其他文件中定义的函数，或者说把该函数的作用域扩展到本文件。例如【思考题 5-10】中的语句：

```
extern input(char str[50]);
/*声明在本函数中将要调用的在其他文件中定义的 3 个函数*/
extern del(char str[],char ch);
extern output(char str[50]);
```

就是说明这 3 个函数为其他文件中定义的外部函数，在该文件中要使用这 3 个函数，所以就要在主函数开始处说明它们。

练一练

【练习 5-7】 分析下列程序运行结果：

源程序如下：

```
int n=1;
main()
{ static int x=5;
  int y;
  y=n;
  printf("MAIN:x=%2d y=%2d n=%2d\n",x,y,n);
  func();
  printf("MAIN:x=%2d y=%2d n=%2d\n",x,y,n);
  func();
}
func()
{ static int x=4;
  int y=10;
  x=x+2;
  n=n+10;
  y=y+n;
  printf("FUNC:x=%2d y=%2d n=%2d\n",x,y,n);
```

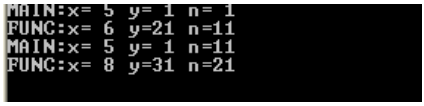
```
}
```

解：程序说明：首先，main 函数中的静态变量 x 被赋初值为 5，y 赋值为全局变量 n 的值 1。输出 3 个变量，结果为：x=5,y=1,n=1。

然后调用 func()函数，在该函数中，局部静态变量 x 被赋值为 4；y 被赋值为 10，然后 x 加上 2 变为 6；n 为全局变量，在主函数中为 1，则现在加上 10 变成 11；y 又被加上 n（为 11）后值为 21。再输出 3 个变量，结果为：x=6，y=21，n=11。

返回到主函数中，输出主函数中的 3 个变量，上次主函数中的 3 个值为 x=5，y=1，n=1。而 x 为局部静态变量，则还为 5；而 y 是局部变量，也是原来值 1；但 n 是全局变量，在 func()函数中变成 11，则回到主函数中也为 11。所以输出结果为：x=5，y=1，n=11。

再次调用 func()函数，局部静态变量 x 上次的值是 6；y 不是静态变量，所以重新被赋值为 10；全局变量 n 值还是主函数中的 11。让 x 的值加上 2 变成了 8，n 的值加上 10 变成了 21，y 的值加上 n 值为 10+21 等于 31。所以输出结果为：x=8，y=31，n=21，程序运行结束。程序运行的结果如下：



本讲小结

这一讲我们学习了变量的作用域和生存期，局部变量、全局变量的定义及其存储类型；函数的作用域，即用 static 修饰的函数只允许被本文件中的函数调用，而没有用 static 修饰的函数则允许同一程序任何文件中的函数使用；以及如何使用 Turbo C 集成环境运行多个 C 语言源程序。

想一想

在编写程序时如何使用函数？使用函数有什么好处？什么是全局变量？什么是局部变量？

本章小结

本章着重介绍了函数的定义、函数的调用和函数原型；递归函数的定义及调用；函数与函数之间的数据传递。同时还介绍了模块化的程序设计及 C 源程序基本结构；变量及函数的作用域和存储类型。

C 语言函数有两种，一种是由系统提供的标准函数，这种函数用户可以直接使用；另一种是用户自定义的函数，这种函数用户必须先定义后使用。在对函数进行定义时，有返回值的函数必须用“return 表达式;”结束函数的运行；若函数是以“return;”结束运行的，说明该函数是无返回值函数。

函数原型是提供函数调用接口信息的说明形式，其格式就是在函数定义格式的基础上去掉了函数体，可见函数定义涵盖函数原型，同样能提供有关的接口信息。

函数可以作为表达式调用，也可以作为语句调用。函数调用时通常以传值的方式传递参数，改动形参变量的值不会影响对应实参变量。定义于函数外的变量称为全局变量，用 static 修饰的全局变量只允许被本文件中的函数访问，而没有用 static 修饰的全局变量则允许同一程序任何文件中的函数访问。定义于函数内的变量称为局部变量，只允许定义该变量的复合语句内的语句使用。用 static 修饰的局部变量称为静态局部变量，可用于在本次调用与下次调用之间传递数据。

用 static 修饰的函数只允许被本文件中的函数调用，而没有用 static 修饰的函数则允许同一程序任何文件中的函数调用。

课后习题五

一、选择题

1．以下正确的函数定义形式是_____。

A）double fun(int x,int y)　　B）double fun(int x;int y)

C）doulle fun(int x,int y);　　D）double fun(int x,y);

2．若有程序段，则以下叙述中不正确的是_____。

```
void f(int n);
main() { void f(int n); f(5);}
void f(int n) { printf("%d",n); }
```

A）若只在主函数中对函数 f 进行声明，则只能在主函数中正确调用函数 f

B）若在主函数前面对函数 f 进行声明，则在其后的函数中都可以正确调用函数 f

C）对于以上程序，编译时系统会提示出错信息，对 f 函数重复声明

D）函数 f 无返回值，所以可用 void 定义 f 的函数类型

3．C 语言规定，简单变量作实参时，它和对应形参之间的数据传递方式是_____。

A）地址传递　　B）单向值传递

C）由实参传递给形参，再由形参传回给实参　　D）由用户指定传递方式

4．C 语言允许函数值类型缺省定义，此时该函数值隐含的类型是_____。

A）float 型　　B）int 型　　C）long 型　　D）double 型

5．以下说法正确的是，如果在一个函数中的复合语句中定义了一个变量，则该变量_____ 。

A）只在该复合语句中有效　　B）在该函数中有效

C）在本程序范围内均有效　　D）为非法变量

6．以下程序的输出结果是_____ 。

```
#include "stdio.h"
int f(int b[],int n)
{ int i,r=1;
  for(i=0;i<=n;i++)
    r=r*b[i];
  return r;
}
main()
{ int x,a[]={2,3,4,5,6,7,8,9};
  x=f(a,3);
  printf("%d",x);
}
```

A）720　　B）120　　C）24　　D）6

7．以下程序的输出结果是_____。

```
#include "stdio.h"
void fun(int a,int b,int c)
{ c=a*b;}
main()
{ int c;
  fun(2,3,c);
  printf("%d",c);
}
```

A）0　　B）1　　C）6　　D）无定值

8．以下程序的输出结果是_____。

```
#include "stdio.h"
int f(int n)
{ if(n==1)
    return 1;
  else
    return f(n-1)+1;
}
main()
{ int i,j=0;
  for(i=1;i<3;i++)
    j+=f(i);
  printf("%d",j);
}
```

A）4　　B）3　　C）2　　D）1

9．若调用一个函数，且此函数中没有 return 语句，则正确的说法是该函数_____。

A）没有返回值

B）返回若干个系统默认值

C）能返回一个用户所希望的函数值

D）返回一个不确定的值

10．下面函数调用语句含有实参的个数为_____。

```
func((exp1,exp2),(exp3,exp4,exp5));
```

A）1　　B）2　　C）4　　D）5

11．在 C 语言程序中，以下正确的描述是_____。

A）函数的定义可以嵌套，但函数的调用不可以嵌套

B）函数的定义不可以嵌套，但函数的调用可以嵌套

C）函数的定义和函数的调用均不可以嵌套

D）函数的定义和函数的调用均可以嵌套

12．以下程序的输出结果是_____。

```
#include "stdio.h"
int func(int a,int b)
{ return(a+b); }
main()
{ int x=2,y=5,z=8,r;
  r=func(func(x,y),z);
  printf("%d",r);
}
```

A）12　　B）13　　C）14　　D）15

13．以下程序的输出结果是_____。

```
#include "stdio.h"
char fun(char ch) {if(ch>='A'&&ch<='Z') ch=ch-'A'+'a'; return(ch);}
main()
{ char s[]="CHINA+abc=defDEF",i=0;
  while(s[i]) {s[i]=fun(s[i]);i++;}
  printf("%s",s);
```

```
}
```

A）CHINA+abc=DEFdef　　B）CHINA+abc=defdef

C）chinaABCDEFdef　　D）china+abc=defdef

14．以下程序的输出结果是_____。

```
void  f(int  b[])
{  int i;
   for(i=2;i<6;i++)   b[i]*=2;
}
main()
{  int  a[10]={1,2,3,4,5,6,7,8,9,10},i;
   f(a);
   for(i=0;i<10;i++)
      printf("%d,",a[i]);
}
```

A）1，2，3，4，5，6，7，8，9，10，　　B）1，2，6，8，10，12，7，8，9，10，

C）1，2，3，4，10，12，14，16，9，10，　　D）1，2，6，8，10，12，14，16，9，10，

15．若用数组名作为函数调用的实参，传递给形参的是_____。

A）数组的首地址　　B）数组第一个元素的值

C）数组中全部元素的值　　D）数组元素的个数

二、填空题

1．以下程序的运行结果是_____。

```
#include "stdio.h"
void  dd();
int  m[10];
main()
{  int  i;
   for(i=0;i<10;i++)
     m[i]=i;
   dd();
   for(i=0;i<10;i++)
     printf("%d  ",m[i]);
}
void  dd()
{  int  j;
   for(j=0;j<10;j++)
     m[j]=m[j]*10;
}
```

2．以下程序的运行结果是_____。

```
#include  "stdio.h"
main()
{  int  k=4,m=1,p;
   p=func(k,m);
   printf("%d,",p);
   p=func(k,m);
   printf("%d\n",p);
}
func(int a,int b)
{  static  int  m=0,i=2;
```

```
    i+=m+1;
    m=i+a+b;
    return(m);
}
```

3．以下程序的功能是用递归方法计算学生的年龄，已知第 1 位学生的年龄最小，为 10 岁，其余学生一个比一个大 2 岁，求第 5 位学生的年龄。请写出空白处的程序。

递归公式如下：

$$age(n)=\begin{cases}10 & (n=1)\\ age(n-1)+2 & (n>1)\end{cases}$$

```
#include <stdio.h>
age(int n)
{ int c;
  if(n==1)
    c=10;
  else
    c=【1】;
  return(c);
}
main()
{ int n=5;
  printf("age:%d\n",【2】);
}
```

【1】_____ 【2】_____

4．以下程序的运行结果是【1】，其算法是【2】。

```
main()
{ int a[5]={5,10,-7,3,7},i;
  sort(a);
  for(i=0;i<=4;i++)
    printf("%d ",a[i]);
}
sort(int a[])
{ int i,j,t;
  for(i=0;i<4;i++)
    for(j=0;j<4-i;j++)
      if(a[j]>a[j+1])
       { t=a[j];
         a[j]=a[j+1];
         a[j+1]=t;
       }
}
```

【1】_____ 【2】_____

5．以下程序的运行结果是_____。

```
main()
{ increment();
  increment();
  increment();
}
increment()
{ static int x=0;
```

```
    x+=1;
    printf("%d",x);
}
```

三、编程题

1．利用递归函数 xpower(x,n)计算 x 的 n 次方的值，其中 n 为非负整数。

2．已有变量定义和函数调用语句：int a=1,b=-5,c;c=fun(a,b);。fun 函数的作用是计算两个数之差的绝对值，并将差值返回调用函数。

3．设计一个程序，输入某年某月某日，判断这一天是这一年的第几天。

第六章　编译预处理

C 程序的编译可分成编译预处理和正式编译两个步骤。在编译 C 源程序时，系统将自动调用编译预处理程序，根据编译预处理命令对程序进行适当的加工，处理完毕自动进入对源程序的正式编译。

编译预处理命令主要有三种，即宏定义、文件包含和条件编译。

编译预处理命令是由 ANSI C 统一规定的，但是它不属于 C 语言语句的范畴，所使用的命令单词也不是 C 语言的保留字。为了表示区别，源程序中的预处理命令均以“#”开头，结束不加分号，以区别源程序中的语句，它们可以写在程序中的任何位置，作用域是自出现点到源程序的末尾。

第十四讲　宏定义、文件包含和条件编译



学习目标

- 掌握不带参数的宏定义的格式及使用方法;
- 掌握带参数的宏定义的格式及使用方法;
- 掌握文件包含的格式及使用方法;
- 掌握条件编译的格式及使用方法。

宏定义是用一个指定的标识符（又称宏名）定义为一个字符串（又称宏体）。在编译预处理时，对程序中所有在宏定义中定义的标识符，都用宏定义中的相应字符串替换，称为“宏替换”或“宏展开”。

C 语言的宏定义分为两种：一种是简单宏定义，即不带参数的宏定义；另一种是复杂的宏定义，即带参数的宏定义（有参宏定义）。

一、不带参数的宏定义

【思考题 6-1】　输入圆的半径，求圆的周长、面积（要求使用无参宏定义圆周率）。



程序分析

在程序中要用到无参宏定义，要注意例程开头的宏定义语句的书写格式，掌握程序在编译之前怎样替换该宏？



编写程序代码

```
#define  PI  3.1415926            /*PI 是宏名，3.1415926 为用来替换宏名的常数*/
main()
 {  float  radius,length,area;
    printf("Input  a  radius: ");
    scanf("%f",&radius);
    length=2*PI*radius;           /*引用无参宏求周长*/
```

```
    area=PI*radius*radius;          /*引用无参宏求面积*/
    printf("length=%.2f,area=%.2f\n",length,area);
}
```

调试程序及运行结果

程序中输入圆的半径值 2.4，程序的运行结果如下：

```
Input  a  radius: 2.4
length=15.08,area=18.10
```

小讲堂

1．不带参数的宏定义

（1）不带参数宏定义的一般形式如下：

```
#define  宏名  字符串
```

其中，#define 是宏定义的命令，标识符和字符串之间用空格分开。标识符称为“宏名”，通常使用大写英文字母和有直观意义的标识符命名，以区别于源程序中的其他标识符。字符串又称为宏体，一般是常数、关键字、语句、表达式，也可以是空白。

（2）功能。不带参数的宏定义的功能是将宏体字符串符号化。在程序中凡出现该宏名的位置，经编译预处理的加工，都被替换成对应的宏体字符串，称之为“宏展开”。

注意 由于宏定义不是 C 语句，不必也不能在行末加分号。

例如，在【思考题 6-1】的例程中，宏定义的语句：

```
#define  PI  3.1415926
```

其中，PI 为宏定义的宏名，3.1415926 为宏定义的宏体。在编译之前，主函数中所有 PI 都会先用 3.1415926 来替换，然后程序再进行编译。使用该宏定义的好处是可以减少出错，如果 PI 的精度进行改变时，只需在本语句中将宏体的值 3.1415926 进行改变即可（例如可将 3.1415926 改为 3.14，则程序中所有 PI 都会用 3.14 来替换，然后程序再进行编译）。

小讲堂

2．关于宏的几点说明

（1）使用宏名代替一个字符串，可以减少程序中重复书写某些字符串的工作量，增加程序的可读性，而且不易出错。

（2）编译预处理时，将程序中 PI 用 3.1415926 代替，与宏调用的过程相反，这种将宏名替换成字符串的过程称为“宏展开”。

（3）C 语言中，用宏名替换一个字符串是简单的转换过程，不作语法检查。若将宏体的字符串中符号写错了，宏展开时照样代入，只有在编译宏展开后的源程序时才会提示语法错误。例如：

```
#define  PI  3.1415926
```

若把字母 6 写成 B，即：

```
#define  PI  3.141592B
```

预处理时照样替换，而不管其含义是否正确，一直到对宏展开的结果进行编译时，才会产生

错误提示。

（4）一个宏名只能被定义一次，否则会出现重复定义的错误。

（5）宏定义可以嵌套。在宏体中，可以出现已定义的宏名，例如：

```
#define  PI  3.1415926
#define  PIR  (PI*r)          /*PI 为已定义的宏名*/
```

（6）如果宏定义一行书写不下，可用反斜线“\”和回车键来结束本行，然后在下一行继续书写。例如，有如下程序：

```
#define  STR  "Hello,\
all the world people!"
main()
{ printf("%s\n",STR); }
```

在编译预处理时，printf 语句中的 STR 将被替换为：

```
"Hello, All the world people!"
```

运行程序将输出：

```
Hello,all the world people!
```

（7）程序中出现的由双引号括起来的字符串，即使和宏名相同，也不进行宏替换。例如，在输出函数 printf()中如果在双引号内有与宏名相同的字符串，也不认为是宏，只认为是普通字符串原样输出。

（8）宏定义命令行放在源程序的函数外时，宏名的作用域从宏定义命令行开始到本源文件结束。

（9）宏名的作用域可以使用#undef 命令终止，形式如下：

```
#undef    标识符
```

则在#define 语句定义了该宏之后，到#undef 命令之前的程序中，该宏定义都有效，但在#undef 命令后该宏则无效了。

二、带参数的宏定义

【思考题 6-2】 求两个数 a 和 b 中的最大值，要求用带参数的宏定义来实现。



程序分析

这个程序可以使用函数来实现，但这里我们要求用带参数的宏定义来实现该功能。要注意带参数的宏定义的格式及在编译前进行替换的方式。



编写程序代码

```
#define  MAX(a,b)  ((a)>(b)?(a):(b))
/*MAX 为带参数的宏定义，a、b 为其参数*/
main()
{  int  a,b,max;
   printf("\nPlease input 2 integer number:");
   scanf("%d%d",&a,&b);
   max=MAX(a,b);     /*求 a 和 b 中的最大数，MAX 为带参数的宏定义*/
   printf("the max value is :%d",max);
}
```



调试程序及运行结果

程序输入两个整数 3 和 5，程序运行结果如下：

```
Please input 2 integer number:3 5
the max value is :5_
```



3．带参数的宏定义

1）带参数宏定义的一般格式

```
#define  宏名（形式参数表）  字符串
```

2）带参数的宏的调用和展开

（1）带参数的宏的调用格式如下

```
宏名（实参表）
```

（2）带参数的宏展开。带参数的宏展开是用宏调用提供的实参字符串，直接置换宏定义命令行中相应形参字符串，非形参字符保持不变。

带参数的宏展开时要注意以下两点：

① 带参数的宏展开是按#define 命令行中指定的字符串从左到右进行置换。

② 如果宏体字符串中包含宏名中的形参，则将程序语句中相应的实参代替形参，如果字符串中的字符不是参数字符，则保留。



4．关于带参数的宏定义的几点说明

（1）定义有参数的宏时，宏名应当与参数表的左括号紧紧相连。否则，C 编译系统将空格以后的所有字符均作为替代字符串，而将该宏视为无参宏。

（2）宏定义时，应将整个字符串以及其中的各个参数均用圆括号括起来，以确保宏展开后字符串中各个参数的计算顺序的正确性，避免出现错误。例如，宏定义为：

```
#define  S(a,b)  a*b
```

在程序中遇到如下语句：

```
m=S(a+1,b+1)*c;
```

对其进行宏展开如下：

```
m=a+1*b+1*c;
```

此时表达式变为 a+b+c，这与我们想要的((a+1)*(b+1))*c 不同，所以出错。可将每个参数和整个字符串都用括号括起来，改为以下宏定义：

```
#define  S(a,b)  ((a)*(b))
```

再对以上语句进行宏展开，结果如下：

```
m=((a+1)*(b+1))*c;
```

这正是我们想要的结果。

（3）在宏定义中的形参是标识符，而宏展开的实参可以是表达式。例如上面的语句：

```
m=S(a+1,b+1)*c;
```

在宏调用的语句 S(a+1,b+1)中表达式 a+1 和 b+1 为 S 的两个实参。

（4）虽然有参数的宏与有参数的函数确实有相似之处，但不同之处更多，主要有以下几个方面。

① 调用有参函数时，是先求出实参的值，然后再复制一份给形参。而展开有参宏时，只是将实参简单地置换形参。

② 在有参函数中，形参是有类型的，所以要求实参的类型与其一致；而在有参宏中，形参是没有类型信息的，因此用于置换的实参，什么类型都可以。有时，可利用有参宏的这一特性，实现通用函数的功能。

③ 函数调用是在程序运行中处理的，临时给它分配存储单元。而宏代换是在编译时进行的，并不给它分配存储单元，不进行数值的传递，也没有返回值。

④ 函数的形参和实参都要求定义类型，而且二者要求一致，若二者不一致时，还要进行类型转换。而对于宏则不存在此类问题，宏没有类型，参数也没有类型，它们都仅是一种符号代表，宏展开时只是进行对应符号的代换。

⑤ 在程序中使用宏时，宏展开后源程序会变长。而函数调用不会使源程序变长。

⑥ 宏代换不占程序运行时间，只占编译时间。而函数调用则占运行时间（分配单元、保留现场、值传递、返回）。

三、文件包含处理

【思考题 6-3】 修改【思考题 6-2】，要求将【思考题 6-2】中的带参数宏定义语句单独放在一个头文件中，主函数单独放在一个源程序文件中，功能同【思考题 6-2】。



程序分析

将一部分内容放到一个头文件中，在编译预处理时，将头文件内容先替换到文件包含语句部分，然后再进行编译。注意头文件和源程序的组成。



编写程序代码

（1）in6_3.h 的内容：

```
#define  MAX(a,b)  ((a)>(b)?(a):(b))
/*MAX 为带参数的宏定义，a、b 为其参数*/
```

（2）主程序内容：

```
#include  "in6_3.h"
/*文件包含命令，将 in6_3.h 头文件包含到该程序中*/
main()
{  int  a,b,max;
   printf("\nPlease input 2 integer number:");
   scanf("%d%d",&a,&b);
   max=MAX(a,b);     /*求 a 和 b 中的最大数，MAX 为带参数的宏定义*/
   printf("the max value is :%d",max);
}
```



调试程序及运行结果

程序运行结果同【思考题 6-2】的例程结果。

注意 在运行主程序之前要将这两个文件都复制到 TC 的目录中，再运行主程序即可。如果不想复制到 TC 的目录下，可以在文件包含命令中加上该文件的路径，例如，文件 in6_3.h 在 C 盘根目录下，路径为"c:\in6_3.h"，则可将文件包含命令改为：

```
#include  "c:\in6_3.h"
```



5．文件包含处理

（1）编译预处理的文件包含功能。

① 编译预处理的文件包含功能是一个源程序通过#include 命令把另外一个源文件的全部内容嵌入到源程序中来。

② 编译预处理程序把#include 命令行中所指定的源文件的全部内容放到源程序的#include 命令行所在的位置。

③ 在编译时并不是作为两个文件连接，而是作为一个源程序编译，得到一个目标文件。

（2）文件包含#include 命令格式主要有以下两种。

① 只检索 C 语言编译系统所确定的标准目录，格式如下：

```
#include   <文件名>
```

② 首先对使用包含文件的源文件所在的目录进行检索，若没有找到指定的文件，再在标准目录中检索，格式如下：

```
#include   "文件名"
```

（3）#include 命令的嵌套使用。当一个程序中使用#include 命令嵌入一个指定的包含文件时，被嵌入的文件中还可以使用#include 命令，又可以包含另外一个指定的包含文件。例如，有三个 C 的源程序文件 f1.c 文件、f2.c 文件和 f3.c 文件。

```
f1.c 文件：
#include  "f2.c"
void  fu1( )
{
   …
}
```

```
f2.c 文件：
  void  fu2( )
{
    …
}
```

```
f3.c 文件：
   #include  "f1.c"
void main( )
  {
    …
  }
```

如果想在 f1.c 文件中使用 f2.c 文件，只要在 f1.c 文件的头部增加一条语句：

```
#include  "f2.c"
```

就可以将 f2.c 中的内容增加到 f1.c 文件中。同理也可在 f3.c 文件包含 f1.c 文件。

（4）文件包含的优点。一个大程序，通常分为多个模块，并由多个程序员分别编程。有了文件包含处理功能，就可以将多个模块共用的数据（如符号常量和数据结构）或函数，集中到一个单独的文件中。这样，凡是要使用其中数据或调用其中函数的程序员，只要使用文件包含处理功能，将所需文件包含进来即可，不必再重复定义它们，从而减少重复劳动。

（5）关于预处理的几点说明。

① 编译预处理时，预处理程序将查找指定的被包含文件，并将其复制到#include 命令出现的位置上。

② 常用在文件头部的被包含文件，称为“标题文件”或“头部文件”，常以“h”（head）

作为后缀，简称头文件。在头文件中，除可包含宏定义外，还可包含外部变量定义、结构类型定义等。例如，【思考题 6-3】主程序中的命令：

```
#include "exam17_3.h"
```

就是在编译之前将 exam17_3.h 文件包含到主程序该命令所在行。

③ 一条包含命令，只能指定一个被包含文件。如果要包含 n 个文件，则要用 n 条包含命令。

④ 文件包含可以嵌套，即被包含文件中又包含另一个文件。

四、条件编译

【思考题 6-4】 输入一行字母字符，根据需要设置条件编译，使之能将字母全改为大写输出，或全改为小写字母输出。

程序分析

定义一个常量 LETTER，通过判断 LETTER 的值来对某些程序进行条件编译。当 LETTER 已经定义过时将字符串 str 中的全部字符转换成大写字母，没有定义过 LETTER 时将大写字母转换成小些字母。注意条件编译的语句格式。

编写程序代码

```
#include "string.h"
#define LETTER 1
main()
{ char str[100], c;
  int i;
  i=0;
  printf("\nPlease input a string:\n");
  gets(str);            /*注意：如果要输入带空格的字符串必须用 gets 函数*/
  while((c=str[i])!='\0')
  { i++;
    #ifdef LETTER
      if(c>='a' && c<='z')
        c=c-32;
    #else
      if(c>='A' && c<='Z')
        c=c+32;
    #endif
    printf("%c", c);
  }
}
```

调试程序及运行结果

输入一个字符串"Hello,all the world people!"，程序将这一串小写字符变成大写字符"HELLO,ALL THE WORLD PEOPLE!"。程序运行结果如下：

```
Please input a string:
Hello,all the world people!
HELLO,ALL THE WORLD PEOPLE!_
```

小讲堂

6．条件编译

所谓条件编译，是指对源程序进行选择性编译。通常情况下，C 语言程序的所有程序行都需进行编译，但有时可能希望程序的某个程序段在满足一定条件时才决定进行编译或不进行编译。使用条件编译功能，为程序的调试和移植提供了有力的机制，使程序可以适应不同系统和硬件设置的通用性和灵活性。

1）#ifdef 命令（或#ifndef 命令）

（1）#ifdef 命令的一般格式如下

```
#ifdef   标识符
      程序段 1
[#else
      程序段 2]
#endif
```

（2）功能。#ifdef 命令的功能是如果“标识符”已经被#define 命令定义过，则编译程序段 1，否则编译程序段 2。

注意

- 在不同的系统中，一个 int 型数据占用的内存字节数可能是不同的。
- 利用条件编译，还可使同一源程序既适合于调试（进行程序跟踪、打印较多的状态或错误信息），又适合高效执行要求。

#ifndef 命令格式与#ifdef 命令一样，功能正好与之相反。

2）#if 命令

（2）#if 命令一般格式如下：

```
#if    常量表达式
      程序段 1
[#else
      程序段 2]
#endif
```

（2）功能。#if 命令的功能是当表达式为非 0（“逻辑真”）时，编译程序段 1，否则编译程序段 2。

（3）条件编译和 if 语句的区别。

① if 语句控制某些语句是否被执行，#if 命令控制着某个程序段是否被编译。

② 用 if 语句调试程序成功后，其调试语句仍被编译成目标代码，只是不再执行，成为废码。而使用条件编译调试程序成功后，其调试语句不再被编译，不生成目标代码，没有废码产生，空间借用率较高。

练一练

【练习 6-1】 设计一个程序，从 3 个数中找最大数，用带参数的宏定义实现。

解：因为条件语句可以通过一条语句实现求 3 个数中最大值的功能，所以可以设一个带 3 个参数的宏定义，宏的值用一个条件语句来实现。



注意 在宏体部分的每一个参数和整个宏体都要用圆括号括起来，以防止宏展开时出错。

源程序如下：

```
main()
{  int  sum,i;
   sum=0;
   for(i=1;i<=100;i++)
   {  if(i%2==0)
        continue;
      sum+=i;
   }
   printf("\n1+3+5+…+99=%d\n",sum);
}
```

程序运行结果如下：

```
Please input 3 int number:5 2 9
The max number is:9
```

【练习 6-2】 用条件编译方法实现以下功能：输入一行电报文字，任选两种输出方式，一为原文输出，二为将字母变成其下一个字母（如'a'变成'b'，'b'变成'c'，依次类推，最后'z'变成'a'，其他字符不变）。用#define 命令来控制是否要译成密码，例如：

```
#define  CHANGE  1
```

则输出密码。若：

```
#define  CHANGE  0
```

则不译成密码，按原文输出。

解：当字母不为'z'或'Z'时，将字母变成其下一个字母，就是将字母的 ASCII 码值加 1 即可实现；当字母为'z'或'Z'时，令该字母等于'a'或'A'。条件编译可以用#ifdef 命令来实现。源程序如下：

```
#include  "string.h"
#define  CHANGE  1    /*可将 1 改为 0，则进行字母替换*/
main()
{  char  str[100],ch;
   int  i,len;
   printf("\nPlease input a string:");
   gets(str);
   len=strlen(str);
   #if  CHANGE
      for(i=0;i<len;i++)
         if(str[i]>= 'a'&&str[i]<'z'|| str[i]>= 'A'&&str[i]<'Z')
            str[i]=str[i]+1;
         else
         {   if(str[i]== 'z')
               str[i]= 'a';
            else if(str[i]== 'Z')
               str[i]= 'Z';
         }
    #endif
    puts(str);
```

```
}
```

程序运行结果如下：

```
Please input a string:Hello,all the world people!
Ifmmp,bmm uif xpsme qfpqmf!
```

想一想

编译预处理命令一般使用在哪里？什么情况要用到编译预处理命令？编译预处理命令有哪几种？各有什么功能？

本章小结

C 程序的编译可分为编译预处理和正式编译两个步骤。编译预处理是在将源程序生成目标文件之前，对源程序的加工。正确使用编译预处理命令可以有效提高程序的开发效率。

C 语言提供了多种预处理命令，常用的有宏定义、文件包含和条件编译。

宏定义命令为#define，是用一个标识符来代表一个字符串，这个字符串可以是常量、变量或表达式。在宏展开中将用该字符串代替宏名。宏定义还可以带参数，宏展开时除用字符串代替宏名外，还应用实参代替形参。

文件包含命令为#include，它可以把一个或多个文件嵌入到另一个源文件中进行编译，结果将生成一个目标文件。

条件编译根据条件成立与否，选择编译程序中满足条件的程序段，便于程序调试，并使生成的目标程序较短，减少内存开销，提高程序运行效率。

课后习题六

一、选择题

1．下列语句中，正确的是_____。

A）#define　MAX=3　　B）#include　math.h

C）#define　max=6　　D）#include　<math.h>

2．在宏定义#define　PI　3.14159 中，用宏名 PI 代替了一个_____。

A）常量　　B）单精度数　　C）双精度数　　D）字符串

3．设有如下宏定义，执行语句 z=2*(N+Y(5+1));后，z 的值为_____。

```
#define  N  3
#define  Y(n)  ((N+1)*(n))
```

A）48　　B）54　　C）出错　　D）42

4．以下叙述中正确的是_____。

A）预处理命令行都必须以“#”开头

B）在程序的一行上可以出现多个有效的预处理命令行

C）使用带参数的宏定义时，参数的类型应与宏定义时的一致

D）以下是正确的宏定义：#define　IBM-PC。

5．以下关于文件包含命令的说法正确的是_____。

A）文件包含命令为#define

B）一个源程序中只能有一条文件包含命令

C）编译含有文件包含命令的源程序时会产生多个目标文件

D）文件包含命令有两种格式

6．下列叙述中正确的是_____。

A）宏名必须用大写字母表示

B）引用带参数的宏时，实参类型应与宏定义中的形参类型一致

C）在程序的一行上可以出现多个有效的宏定义

D）宏展开不占用运行时间，只占用编译时间

7．以下程序的运行结果为_____。

```
#define  ADD(x)  x+x
main()
{  int  a=1,b=2,i=3;
   int  sum;
   sum=ADD(a+b)*i;
   printf("sum=%d",sum);
}
```

A）sum=18　　B）sum=60　　C）sum=10　　D）sum=12

8．设有以下宏定义:

```
#define N 3
#define Y(n) ((N+1)*n)
```

则执行语句 z=2*(N+Y(5+1));后，z 的值为_____。

A）出错　　B）42　　C）48　　D）54

9．下列程序执行后的输出结果是_____。

```
#define  MA(x)  x*(x-1)
main()
{  int  a=1,b=2;
printf("%d\n",MA(1+a+b));
}
```

A）6　　B）8　　C）10　　D）12

10．若有如下宏定义：

```
#define  MOD(x,y)  x%y
```

则执行以下程序段后，输出结果为_____。

```
int  z,a=15,b=100;
z=MOD(b,a);
printf("%d",z++);
```

A）10　　B）11　　C）4　　D）宏定义错误

二、填空题

1．以下程序的功能是输出 x 和 y 的乘积，在程序中填入应包含的编译预处理命令_____。

```
main()
{  int  x=2,y=4;
   printf("%d\n",POW(x,y));
}
```

2．设有以下宏定义：

```
#define  WIDTH   80
#define  LENGTH   WIDTH+40
```

则执行赋值语句：

```
v=LENGTH*20;
```

（v 为 int 型变量）后，v 的值是_____。

3．下面程序的运行结果为_____。

```
#define  MUL(z)  (z)*(z)
main()
{
    printf("%d\n",MUL(1+2)+3);
}
```

4．下面程序的运行结果是_____。

```
#define  EXCH(a,b)  { int  t; t=a; a=b; b=t; }
main()
{  int  x=5,y=9;
   EXCH(x,y);
   printf("x=%d,y=%d\n",x,y);
}
```

5．下面程序的运行结果是_____。

```
#define  A  4
#define  B(x)  A*x/2
main()
{  float  c,a=4.5;
   c=B(a);
   printf("%5.1f\n",c);
}
```

三、编程题

1．输入两个整数，求它们相除的余数，用带参数的宏来编程实现。

2．试定义一个带参数的宏 swap(x,y)，以实现两个整数之间的交换，并利用它们将一维数组 a 和 b 的值进行交换（假设数组中都是十个元素）。

3．设计一个程序，判断给定的年份 year 是否为闰年，用带参数的宏定义实现。

4．从键盘输入一行字符，按回车键结束输入。由条件编译求其中大写字母的个数或小写字母的个数并显示。

第七章　指　　针

在前面的章节中，对数据的存取基本上都是通过变量名进行的，没有直接对地址进行操作，但事实上每个变量名都与一个唯一的内存地址相对应。在本章中，我们将要学习一种特殊的专门用于存储地址的变量——指针，通过指针实现间接访问指针所指的数据。

指针在 C 语言中占有重要的地位，同时也是 C 语言的一大特色。C 语言的高度灵活性及其特别强的表达能力，在很大程度上来自于巧妙而恰当地使用指针。C 语言的指针既可以指向各种类型的变量，也可以指向函数。正确地使用指针，能够有效地表示和处理复杂的数据结构，特别是对动态数据的管理。

第十五讲　　指针概述与指针赋值、指针的运算



学习目标

- 理解和掌握指针的基本概念;
- 熟练掌握各种类型指针的定义和初始化、赋值;
- 掌握指针的基本运算;
- 指针的动态申请/释放内存。

一、指针概述与指针赋值

指针是 C 语言的一大特色，使用指针可以灵活而有效地解决许多问题，可以使程序更加简洁和高效。只有深入地理解指针，才能学习和掌握 C 语言的精华。

【思考题 7-1】　阅读下面的程序片断，分析程序功能，并思考实现同样的功能，是否还有其他的方法？

```
main()
{  int  a,b=1;
char  c='a';
a=2;
c='c';
}
```



程序分析

这段程序代码定义了两个整数变量 a 和 b、一个字符型变量 c，其中变量 b 是在定义的同时进行初始化，而变量 a 和 c 则是在定义后单独初始化。

在学习本讲之前，我们对数据的存取基本上都是通过变量名进行的。事实上，在程序运行时，这些数据都是存储在内存中的，都被分配了以字节为单位的顺序存储空间，通过变量名与分配的存储空间的首地址相对应。所分配的存储空间的大小是由变量的类型决定的。

例如，int　a,b=1;，定义了一个整型变量，该变量被分配了 2 个字节的存储空间。而语句，char　c='a';，则定义了一个字符变量，该变量被分配了 1 个字节的存储空间，并存储'a'

这个字符。

现在，我们学习一种特殊的专门用于存储地址的变量——指针变量，通过指针变量间接访问指针所指向的数据。

请读者学习下面的程序片断，它同样实现了【思考题 7-1】中所给程序的功能。

编写程序代码

```
main()
{  int  a,b=1;
   int  *pa,*pb;
   char  c='a';
   char  *pc=&c;
   pa=&a;
   pb=&b;
   *pa=2;
   *pc='c';
  }
```

小讲堂

1．与指针相关的一些概念

（1）存储地址。组成内存空间的是顺序排列的以字节为单位的存储单元，将这些存储单元从 0 开始顺序编号，就构成了每个存储单元的地址。每个数据都存放于从某个特定的地址开始的一个或若干个存储单元中，这个特定的地址即为该数据的存储地址，程序中通过变量名与这个存储地址进行对应。

（2）指针变量，是用于存储数据地址的变量，由于地址指明了数据存储的位置，因此指针变量被形象地称为“指针”，该地址中存放的数据也被形象地称为“指针所指向的数据”，如图 7-1 所示。

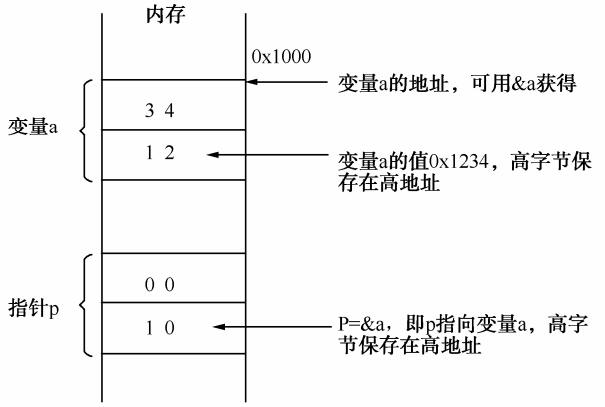


图 7-1　指针及所指向数据示意图

（3）直接访问，指对数据的存取操作通过变量名进行，而没有通过地址进行操作。

（4）间接访问，指通过指针访问指针所指向的数据。

（5）指针的类型。指针是一种变量，因此也具有类型。指针的类型就是指针所指向的数据的类型，比如【思考题 7-1】中指针变量 pa 定义为是指向整型变量的，因此指针 pa 就是一

个整型指针。

（6）指针的长度。指针是一种变量，因此也具有长度。由于指针变量中存储的是一个地址值，而在 C 语言中，地址值是用 16 位二进制数字表示的，因此指针变量的长度是 2 个字节。



2．指针变量的定义及初始化

1）指针变量的定义格式

指针变量的定义格式如下：

```
基类型    *指针变量名;
```

其中，指针变量的命名规则与普通变量的命名规则相同。

“*”是一个说明符，用来说明其后的变量是一个指针变量。格式中的“基类型”用来说明指针的类型，即指针所指向的数据的类型，它必须是一个合法的 C 语言数据类型。如，语句 int a;定义了一个整型变量，而 int *p;定义的是一个指向整型类型数据的指针变量 p。



在 C 语言中，允许使用 void 类型的空类型指针，其定义格式如下：

```
void    *指针变量名;
```

表示指针变量不指向任何一个确定类型的数据，仅仅用来存放一个地址。

2）指针变量的初始化

在定义指针变量的同时，可以给指针一个初值。例如:

```
int   a,*p=&a;
```

指针变量 p 被初始化为指向变量 a（&a 表示变量 a 的地址，&为取地址运算符），变量 a 的定义要出现在 p 之前。



初始化时，必须保证指针所指向的数据类型与指针定义中的数据类型是一致的，因此对空指针（void *）的初始化，需要进行强制类型转换，例如：

```
int  x,*p1=&x;              /* 正确*/
void  *p2=&x;               /* 错误，类型不匹配*/
void  *p2=(void *)&x;       /* 正确*/
void  *p2=(void *)p1;       /* 正确*/
p1=(int *)p2;               /* 正确*/
```



3．指针变量的赋值

一个指针变量除了在定义的同时被赋值（初始化）外，也可以在定义后通过不同的“渠道”获得一个确定的地址值，从而指向一个具体的对象。

（1）通过取地址运算符“&”获得一个变量的地址值。先定义指针变量，然后通过赋值语句给指针一个初值，例如：

```
int  k,*q;
q=&k;
```

注意

①“&”只能用于变量或数组元素，不能用于表达式或常量；

②“&”必须放在运算对象的左边；

③“&”的运算对象必须与指针变量的类型相同，否则需要强制类型转换。

（2）通过数组名给指针变量赋值。

```
int  a[10],*q;
q=a;
```

注意 数组名本身就是一个地址，因此不能加“&”取地址运算符。

（3）通过指针变量获得地址值。

```
int  k,*q,*p;
q=&k;
p=q;
```

注意 赋值运算符两边指针变量的“基类型”必须相同。

（4）通过标准函数获得地址值。通过调用 malloc 和 calloc 函数，在内存中开辟动态存储区域，并把所开辟的动态存储区域的地址赋给指针变量。

```
int   *p1;
p1=(int *)malloc(sizeof(int));
```

注意 用于动态存储分配的函数，返回值的类型都是 void *。在使用时，都要先进行强制类型转换，使之指向一个确定类型的变量。

（5）给指针变量赋 NULL 值。

```
int  *p;
p=NULL;
```

注意

① NULL 是系统定义的符号常量，值为 0。在使用 NULL 时，要在程序的开始处包含：

```
#include  "stdio.h"
```

② 值为 NULL 的指针，是空指针，不能用来访问数据。

③ p=NULL;和 p=0;是等价的，因为 NULL 在 stdio.h 中被定义为常量 0。

二、指针的运算

指针也是一种变量，那么就应该可以作为操作数，和操作符一起组合成表达式，从而实现更复杂的功能。但是，指针是一种特殊的变量，它保存的是地址值，对这样的指针变量，可以参与哪些运算呢？

【思考题 7-2】 假设 p 初值指向 x 的地址，请注意其中指针变量的使用。

程序分析

除了已经学习过的指针的赋值运算，常用的与指针有关的有 3 种运算：指针运算符*、指针的算术运算、指针的关系运算。

请阅读下面的小程序，分析运行结果，并上机验证。

编写程序代码

```
#include  "stdio.h"
main()
{  int  x,*p=&x;
   int  a[5];
   int  *qx=a;
   int  *qy=&a[4];
   printf("%x\t",p);
   printf("%x\t",p++);
   printf("%x\t",++p);
   printf("%x\n",p);
   printf("%x\t",qx);
   printf("%x\n",qx+3);
   printf("%x\t",qy);
   printf("%d\n",qy-qx);
}
```

调试运行程序

程序运行结果如下（假设 p 初值为 0xffc8）：

```
ffc8    ffc8    ffcc    ffcc
ffca    ffd0
ffd2    4
_
```

小讲堂

4．指针运算符

使用指针运算符“*”可以对指针所指向的数据进行间接访问，即对指针所指向的数据进行存取操作。取内容运算的使用格式为：

```
*指针变量
```

表示该指针变量所指向的变量对象。

例如，有如下程序：

```
#include  "stdio.h"
main()
{  int  *p;
   int  i=10,j;
   p=&i;     /* 指针 p 指向变量 i*/
   j=*p;     /* 将指针 p 所指向的地址空间中的整型数据赋值给整型变量 j*/
   *p=20;    /* 对“*p”的操作就相当于对 i 进行操作*/
   printf("%d\t%d\t%d\n",*p,i,j);
}
```

程序的输出结果为：

```
20   20   10
```



5．指针的算术运算

（1）算术运算主要是对变量进行加、减、乘、除的操作，但对指针变量只能进行加、减操作，其他操作是没有意义的，而且一般用在对数组元素进行操作的场合。

（2）四种可以应用于指针的算术运算符：++、−−、+、−。

① 指针的自增或自减运算：p++、++p、p−−、−p，是使指针指向下一个或前一个同类型的数据。运算后指针变量的值取决于它指向的数据类型，即，指针的自增或自减运算都是以数据类型的长度大小作为单位来进行的。

② 指针变量加上或减去一个整数：px+n、px−n，其中，n 是一个正整数。指针作为地址量加上或减去一个正整数 n，其意义是指向当前位置之后或之前第 n 个数据的位置。由于指针可以指向不同数据类型，所以这种运算的结果值取决于指针指向的数据类型。对于某种数据类型的指针 p 来说：

p+n 的实际操作是：(p)+n*sizeof(数据类型)；

p−n 的实际操作是：(p)−n*sizeof(数据类型)。

③ 两个指针变量的减法运算：py−px，两个指向相同数据类型的指针可以进行减法运算，结果是一个整数，表示两个地址之间可以容纳的相同数据类型的数据的个数。它执行的运算不是两指针中的地址值简单相减，而是按照公式：(py−px)/sizeof(数据类型)来计算的。



6．指针的关系运算

（1）在两个指向相同类型变量的指针之间可以进行各种关系运算，用来表示它们指向的地址位置之间的关系。

（2）关系运算符有：>、>=、<、<=、！=、==，可以对两个指针进行大小（地址值高低）的比较。指向后方的指针大于指向前方的指针；如果两个指针相等，则表明它们指向同一个数据。

（3）指针关系运算的重要应用：使用空指针进行数据访问是没有意义的，因此，在进行一些指针运算之前，常常需要先检查指针是否为空，再确定能否进行之后的数据访问操作。常用的判断指针 p 是否为空的语句为：

```
    if(p==NULL)  { … }
或  if(!p)       { … }
```



7．指针的指针

指针可以指向各种基本类型的数据，是否也可以有指向指针类型的数据呢？

我们知道，指针变量也是一种数据，在内存中也有存储地址，因此也可以再定义一个指针变量，是指向这个指针的，简称为“指向指针的指针”。定义一个指向指针数据的指针变量的格式如下：

```
类型标识符    **指针变量名;
```

例如：

```
char  ch, *p1=&ch;
char  **p2=&p1;
```

第一个语句定义了一个字符型的简单变量 ch 和一个字符型指针变量 p1，并且指针变量 p1 是指向字符变量 ch 的。第二语句定义了一个指向字符指针的指针变量 p2，并且 p2 是指向字符指针 p1 的。p1、p2 和 ch 这三个变量的关系如图 7-2 所示。

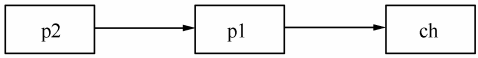


图 7-2　指向指针的指针示意图

练一练

【练习 7-1】　定义一个整型数组 a[4]并初始化，使用指针求出数组中各元素之和。

解：源程序如下：

```
#include  "stdio.h"
main()
{  int  a[4]={1,3,5,7};
    int  *p=a;
    int  sum=0;
    int  i;
    for(i=0; i<4; i++)
    {  sum=sum+(*p);      /* 等价于 sum=sum+a[i];*/
       p++;
    }
    printf("sum=%d\n",sum);
}
```

程序运行结果如下：

```
sum=16
```

【练习 7-2】　编写程序，统计输入字符串的字符个数。

解：源程序如下：

```
#include  "stdio.h"
main()
{  char  str[50];
   char  *p;
   printf("Enter a string(less than 50 characters): ");
   gets(str);
   p=str;
   while(*p!='\0')    p++;
   printf("The length of the string is:   ");
   printf("%d\n", p-str);
}
```

程序运行结果如下：

```
Enter a string(less than 50 characters): glkfh6
The length of the string is:   6
```

【练习 7-3】 通过一个程序来进一步深入学习关于指针自加、自减的前置和后置运算，请读者分析下面程序的输出结果。

```
#include "stdio.h"
main()
{  int  a[5]={1,2,3,4,5};
   int  i;
   int  *pa=a;
   for(i=0;i<5;i++)
     printf("%d\t",a[i]);
   printf("\n");
   printf("%d\n",*pa++);
   printf("%d\n",*++pa);
   printf("%d\n",(*pa)++);
   printf("%d\n",++(*pa));
   for(i=0; i<5; i++)
     printf("%d\t",a[i]);
   printf("\n");
}
```

程序输出结果为：

```
1       2       3       4       5
1
3
3
5
1       2       5       4       5
```

解：程序中修改了数组元素的值，我们一起来分析为什么是这样的输出。

（1）优先级：“*”和“++”属于同级运算，其结合规则是从右至左。

（2）指针 pa 指向数组 a 的首地址，即指向数组中的第一个数据 a[0]。

（3） pa++是后置运算，*pa++等价于*(pa++)，因此表达式先访问的是*pa，输出为 1；之后指针 pa 值加 1，指向数组中的下一个数据，即 a[1]。

（4）表达式*++pa 等价于*（++pa），++pa 是前置运算，因此表达式先将指针 pa 的值加 1，使 pa 指向数组中的下一个数据，即 a[2]，然后才执行输出语句，输出的结果为 3。

（5）表达式(*pa)++也是后置运算，因此先输出指针 pa 指向的数据 a[2]，即数字 3，然后不是将指针 pa 的值加 1，而是将指针 pa 所指向的数据 a[2]的值加 1，a[2]=4。

（6）表达式++(*pa)则是前置运算，首先将指针 pa 所指向的数据 a[2]的值加 1，a[2]=5，然后再输出 a[2]，结果自然为 5。

（7）最后，验证数组 a 中各元素值的变化情况。

本讲小结

本讲学习了各种类型指针的定义、初始化和赋值。重点内容是指针的算术运算和关系运算，要求读者能够熟练掌握并应用。

想一想

通过本讲，读者对指针已经有了初步的了解。本讲的【思考题 7-1】的功能，既可以用指针实现，也可以不使用指针，那么在什么情况下使用指针呢？使用指针有哪些优点呢？

第十六讲　指针与数组（一）

学习目标

- 了解指针与数组的关系；
- 掌握指针在数组方面的应用；
- 能够使用指针引用数组元素。

一个数组可以包含若干个类型相同的元素，每个元素都与一个唯一的地址相对应，可以使用“数组名[下标]”的方法引用数组元素。我们知道，指针变量可以指向简单变量，也可以指向数组元素，那么如何使用指针变量引用数组元素呢？

一、一维数组元素的指针访问方式

【思考题 7-3】　定义一个一维整数数组 a（5 个元素并赋初值），如何分别使用数组名和指针来访问数组中的元素呢？

程序分析

在前面的章节中，我们已经学习过使用“数组名[下标]”的方法访问数组中的指定元素，下面，我们将学习如何通过指针的运算来访问一维数组中的数组元素。

先请读者阅读下面的程序代码，注意程序中使用不同方式访问一维数组的数组元素。

编写程序代码

```
#include "stdio.h"
main()
{  int  i,*p;
   int  a[5]={1,3,5,7,9};
   for(i=0;i<5;i++)
     printf("%d\t",a[i]);
   printf("\n");
   for(i=0;i<5;i++)
     printf("%d\t",*(a+i));
   printf("\n");
   p=a;
   for(i=0;i<5;i++)
     printf("%d\t",*(p+i));
   printf("\n");
   p=a;
   for(i=0;i<5;i++)
     printf("%d\t",p[i]);
   printf("\n");
   p=a;
   for(i=0;i<5;i++)
```

```
    {   printf("%d\t",*p);
        p++;
    }
      printf("\n");
}
```

调试运行程序

```
1       3       5       7       9
1       3       5       7       9
1       3       5       7       9
1       3       5       7       9
1       3       5       7       9
```

小讲堂

1．一维数组元素与元素地址关系

在 C 语言中，一维数组的数组名实际上就是指向数组下标为 0 的元素的指针。

例如，若有如下定义：

```
int   a[5]={1,2,3,4,5},*p;
p=a;
```

则数组 a 中的元素与数组名 a 及指针 p 的关系，如图 7-3 所示。

其中，数组名 a 的类型是 int *，并且指向数组元素 a[0]，即 a 中存放的地址为&a[0]。在前面已经介绍，通过指针运算符“*”可以访问指针所指向的数据，因此可以用*a 来访问元素 a[0]。当然，以这种方式不仅可以访问数组元素 a[0]，还可以访问数组中的其他元素。例如，*(a+1)可以访问 a[1]，*(a+2)可以访问 a[2]，……

一般地说，用*(a+i)可以访问 a[i]，即*(a+i)和 a[i]是完全等价的。

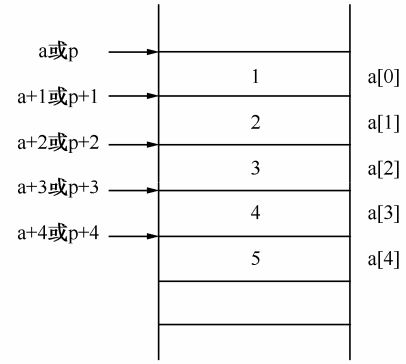


图 7-3　一维数组元素与元素地址关系示意图

同样，指向一维数组首元素的任何指针也可以像一维数组名那样使用。由于指针 p 和数组名 a 均是指向数组 a 的首元素的指针，即 p 和 a 是完全等价的。因此，访问数组 a 中下标为 i 的元素，可以使用如下四种表示形式：a[i]、p[i]、*(a+i)、*(p+i)。



2．一维数组元素的访问

（1）程序实现了使用不同的方式访问数组元素。

（2）我们把通过 a[i]或 p[i]访问数组元素的方式成为下标方式，把通过*(a+i)或*(p+i)访问数组元素的方式称为指针方式。

（3）一维数组中任何一个元素的地址，都可以用其数组名加上一个位移量来表示。这个位移量的单位不是字节，而是数组元素的类型所占的存储空间的大小。

例如，上例中数组元素 a[1]的地址是 a+1，1 并不是代表地址的位移为 1，而是代表 1 个位移单位。对 int 来说，1 个位移单位为 2 个字节，因此 a+1 实际上执行的操作是 a+1*sizeof(int)。

（4）注意：程序中最后一种方式，每次都是使用运算符“*”来访问指针 p 所指向的数据，p 的值需要执行自加运算，从而指向下一个元素。但是，对于数组名 a 来说，在 C 语言中认为是指针常量而不是变量，所以不能改变 a 的值，若写成“a++”是错误的。

二、二维数组元素的指针访问方式

【思考题 7-4】 定义一个二维整型数组 b（9 个元素并赋初值），如何分别使用数组名和指针来访问数组中的元素呢？



程序分析

在 C 语言中，一维数组名代表了该数组的起始地址，实际上是一个指针常量，可以使用下标法和指针法访问数组元素，这样的结论可以推广到二维数组、三维数组中。注意程序中使用不同方式访问二维数组的数组元素。



编写程序代码

```
#include  "stdio.h"
main()
{  int  i,j,*p;
   int  b[3][3]={1,2,3,4,5,6,7,8,9};
   printf("\n");
   for(i=0;i<3;i++)
     for(j=0;j<3;j++)
       printf("%d\t",b[i][j]);
   printf("\n");
   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
        printf("%d\t",*(b[i]+j));
   printf("\n");
   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      printf("%d\t",*(*(b+i)+j));
   printf("\n");
   for(i=0;i<3;i++)
     for(j=0;j<3;j++)
       printf("%d\t",*(&b[0][0]+3*i+j));
   printf("\n");
   for(i=0;i<3;i++)
```

```
        for(j=0;j<3;j++)
          printf("%d\t",*(b[0]+3*i+j));
      printf("\n");
      for(i=0;i<9;i++)
          printf("%d\t",*(b[0]+i));
      printf("\n");
      p=&b[0][0];
      for(i=0;i<9;i++)
          printf("%d\t",*(p+i));
      printf("\n");
      p=&b[0][0];
      for(i=0;i<9;i++)
        printf("%d\t",p[i]);
      printf("\n");
      p=b[0];
      for(i=0;i<9;i++)
      {  printf("%d\t",*p);
         p++;
      }
    printf("\n");
}
```

调试运行程序

```
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9
1       2       3       4       5       6       7       8       9
```

小讲堂

3．二维数组元素的访问

（1）在 C 语言中，可以用一维数组来解释二维数组：二维数组是以一维数组为元素的一维数组。例如，一个二维数组可以定义为：

```
int  b[3][2]={1,2,3,4,5,6};
```

（2）二维数组 b 可以看成是由 b[0]、b[1]、b[2]这 3 个元素组成的一维数组，其中的每个元素本身又是一个具有 2 个元素的 int 型一维数组。一维数组 b[0]、b[1]、b[2]按顺序存储。

（3）二维数组 b 的数组名是指向该二维数组首行的那个一维数组的指针，即数组名 b 指向 b[0]所代表的一维数组。二维数组名 b 可以理解为一个行指针，在表达式 b+1 中，数值 1 的单位应当是 2*2 个字节，而不是 2 个字节。

（4）在二维数组 b 中，b[0]、b[1]、b[2]都是一维数组名，其值依次为二维数组每行首元素的地址，其类型就是数组元素的类型。对于表达式 b[i]+1(0<=i<3)，数值 1 的单位应当是 2 个字节。

（5）b[0]、b[1]、b[2]和 b 都是数组名，因此都代表的是不可变的指针常量，不可以对其值进行修改。

（6）二维数组 b 的元素与数组名、元素地址的关系如图 7-4 所示。

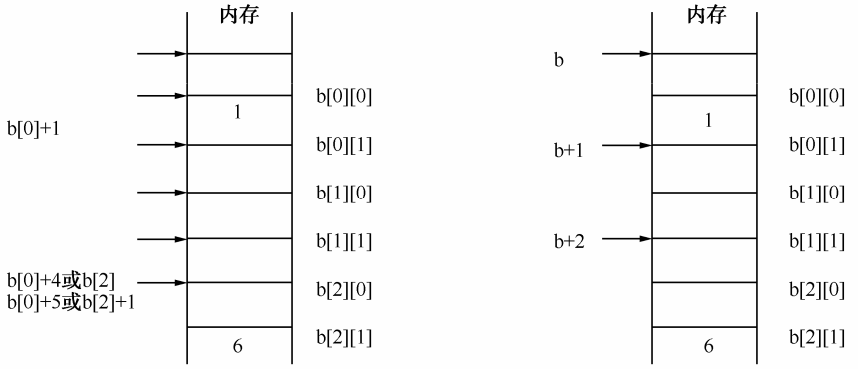


图 7-4　二维数组 b 的元素与数组名、元素地址的关系图

（7）二维数组 b 中 b[i][j]的地址可以用以下五种表达式求得：&b[i][j]、b[i]+j、*(b+i)+j、&b[0][0]+列数*i+j、b[0]+列数*i+j。

三、字符指针与字符串

【思考题 7-5】　如何应用字符指针实现字符串的输入和输出？

程序分析

在 C 语言中，没有字符串类型，是通过字符型数组作为字符串的存储空间的。数组名就是指针，那么任何指向字符型数组首元素的指针都可以代表存储于该处的字符串。

存取一个字符串，可以定义一个字符型数组；也可以定义一个字符型指针，通过指针来访问字符串中的字符。看下面的程序片段：

```
char  str[]="form";
char  *p="abcd",*q;
q=str;
```

（1）直接将一个字符串常量赋给字符指针；

（2）先将一个字符串存放在字符数组中，再将字符数组名赋给字符指针。

编写程序代码

```
#include  "stdio.h"
main()
{  char  str1[]="hello ",str2[10];
   char  *str3,*str4=str1;
   printf("output str1 : ");
   printf("%s\n", str1);
   printf("output str4 : ");
   printf("%s\n", str4);
   printf("input str2 : ");
   gets(str2);
   printf("input str4 : ");
   scanf("%s",str4);
   printf("output str2 : ");
   puts(str2);
```

```
    printf("output str4 : ");
    puts(str4);
    str3=str2+5;
    printf("output str3 : ");
    puts(str3);
}
```


调试运行程序

```
output str1 : hello
output str4 : hello
input str2 : very good
input str4 : very good
output str2 : very good
output str4 : very
output str3 : good
```



4．字符指针与字符串

（1）字符串的输入，格式如下：

```
scanf("%s",str);
```
或
```
gets(str);
```

其中，str 存放的是输入字符串的起始地址。str 可以是字符数组名、字符指针或字符数组元素的地址。


注意 这两个函数是有区别的，scanf()接受从键盘输入的字符串时，遇到空格就认为输入完成，空格后面的输入被认为是下一个参数或无效的。而 gets()接受从键盘输入的字符串时，只有“回车”才被认为输入完成，因此这种输入方式中可以输入含有空格的字符串。

（2）字符串的输出，格式如下：

```
printf("%s",str);
```
或
```
puts(str);
```

其中，str 存放的是输入字符串的起始地址。str 可以是字符数组名、字符指针或字符数组元素的地址。

这两个函数的功能是相同的，输出都是遇到第一个'\0'为止。'\0'是字符串结束标志，不在输出字符之列。

练一练

【练习 7-4】 参考【思考题 7-4】编写程序，定义一个二维整数数组 b[2][4]并初始化，使用不同的方式访问二维数组中的元素。

解：源程序如下：

```
#include  "stdio.h"
main()
{  int  i,j,*p;
   int  b[2][4]={1,2,3,4,5,6,7,8};
   printf("\n");
   for(i=0;i<2;i++)
     for(j=0;j<4;j++)
```

```
        printf("%d\t",b[i][j]);
    printf("\n");
    for(i=0;i<2;i++)
      for(j=0;j<4;j++)
        printf("%d\t",*(b[i]+j));
    printf("\n");
    for(i=0;i<2;i++)
      for(j=0;j<4;j++)
        printf("%d\t",*(*(b+i)+j));
    printf("\n");
    for(i=0;i<2;i++)
      for(j=0;j<4;j++)
        printf("%d\t",*(&b[0][0]+4*i+j));
    printf("\n");
    for(i=0;i<2;i++)
      for(j=0;j<4;j++)
        printf("%d\t",*(b[0]+4*i+j));
    printf("\n");
    for(i=0;i<8;i++)
      printf("%d\t",*(b[0]+i));
    printf("\n");
    p=&b[0][0];
    for(i=0;i<8;i++)
      printf("%d\t",*(p+i));
    printf("\n");
    p=&b[0][0];
    for(i=0;i<8;i++)
      printf("%d\t",p[i]);
    printf("\n");
    p=b[0];
    for(i=0;i<8;i++)
    {  printf("%d\t",*p);
       p++;
    }
    printf("\n");
}
```

```
1       2       3       4       5       6       7       8
1       2       3       4       5       6       7       8
1       2       3       4       5       6       7       8
1       2       3       4       5       6       7       8
1       2       3       4       5       6       7       8
1       2       3       4       5       6       7       8
1       2       3       4       5       6       7       8
1       2       3       4       5       6       7       8
1       2       3       4       5       6       7       8
```

【练习 7-5】 输入一行字符（不超过 40 个），统计其中大写字母、小写字母、数字、空格及其他字符的个数。

解：源程序如下：

```
#include  "string.h"
#include  "stdio.h"
main()
{  char  str[40],*p;
   int  upper,lower,number,space,other;
   upper=0;      /* 存储大写字母个数 */
```

```
        lower=0;      /* 存储小写字母个数 */
        number=0;     /* 存储数字个数 */
        space=0;      /* 存储空格个数 */
        other=0;      /* 存储其他字符个数 */
        printf("pls input characters , no more than 40 :\n");
        gets(str);
        p=str;
        while(*p !='\0')
        {  if(*p>='A' && *p<='Z')            upper++;
           else if(*p>='a' && *p<='z')       lower++;
           else if(*p>='0' && *p<='9')       number++;
           else if(*p == ' ')                space++;
           else                              other++;
           p++;
    }
    printf("\n");
    printf("upper is  :   %d\n",upper);
    printf("lower is  :   %d\n",lower);
    printf("number is :   %d\n",number);
    printf("space is  :   %d\n",space);
    printf("other is  :   %d\n",other);
}
```

```
pls input character , no more than 40 :
ABCD ag 35^ @

upper is  :    4
lower is  :    2
number is :    2
space is  :    3
other is  :    2
_
```

想一想

本讲的难点是如何访问二维数组中的元素，请读者仔细阅读【思考题 7-4】中的各种访问方式，更好地理解 b、b[0]、b[0][0]、b+1、b[0]+1 的关系。

本讲小结

本讲主要学习的是指针与数组的关系，要求读者能够熟练应用并掌握：

（1）使用指针访问数组元素；

（2）应用字符指针访问字符串。

第十七讲　指针与数组（二）



学习目标

- 了解指针与数组的关系，应用数组指针与指针数组；
- 区分数组指针与指针数组。

指针可以指向任何合法的数据类型，如 int、float 等，还可以指向一维数组。指向一维数组的指针不是指向数组中的某一个元素，而是指向由若干个元素组成的一维数组。指向一维

数组的指针也可以称为“行指针”，简称为“数组指针”。

我们知道，指针也是一种具有某种数据类型的变量，也就是说，指针可以作为数组元素。使用相同类型的指针作为数组元素的数组称为“指针数组”。

一、指向一维数组的指针

【思考题 7-6】 输入 3 行字符（不超过 80 个），统计其中大写字母、小写字母、数字、空格及其他字符的个数。要求使用数组指针统计字符串中各种字符信息。



程序分析

在上一讲中，我们已经学习了如何判断一行字符中的大写字母、小写字母、数字、空格及其他字符的个数，使用的是一维数组和指向该数组的指针。现在的要求是 3 行字符，就需要使用二维数组和一个指向二维数组的指针来实现了。

我们定义一个 3 行 80 列的二维数组，存放输入的字符串；再定义一个“行指针”，先指向二维数组中第 0 行。通过这个指针来逐行访问数组中的元素，实现各种字符的统计功能。

先请读者阅读下面的程序代码。



编写程序代码

```
#include  "string.h"
#include  "stdio.h"
main()
{  char  str[3][80],(*p)[80],ch;
   int  i,j;
   int  upper, lower, number, space, other;
   upper=0;      /* 存储大写字母个数 */
   lower=0;      /* 存储小写字母个数 */
   number=0;     /* 存储数字个数 */
   space=0;      /* 存储空格个数 */
   other=0;      /* 存储其他字符个数 */
   printf("pls input characters , no more than 40 per line :\n");
   for(i=0;i<3;i++)
     gets(str[i]);
    p=str;
    for(i=0;i<3;i++)
      for(j=0;j<strlen(str[i]);j++)
      {  ch=*(*(p+i)+j);
       if(ch>='A' && ch<='Z')            upper++;
       else if(ch>='a' && ch<='z')       lower++;
       else if(ch>='0' && ch<='9')       number++;
       else if(ch == ' ')            space++;
       else                          other++;
}
  printf("\n");
  printf("upper is  :   %d\n",upper);
  printf("lower is  :   %d\n",lower);
  printf("number is :   %d\n",number);
  printf("space is  :   %d\n",space);
  printf("other is  :   %d\n",other);
}
```


调试运行程序

程序运行结果如下：

```
pls input characters , no more than 40 per line :
AGAGcga3% 6
ag#56 &  ag
2345$ 6((

upper is  :    4
lower is  :    7
number is :    9
space is  :    5
other is  :    6
```


小讲堂

1．指向一维数组的指针的定义及赋值

程序中有如下的语句，定义了一个指向一维数组的指针，并赋值使之指向一个二维数组的首行。

```
char  str[3][80],(*p)[80];
p=str;
```

（1）定义格式如下：

类型标识符　(*指针名)[常量表达式];

其中，类型标识符是说明指针所指的一维数组中元素的类型，常量表达式说明指针指向的一维数组中元素的个数，如语句 char　(*p)[80]; 声明了一个指向有 80 个字符型元素的一维数组的指针。

（2）指针赋值。一般都是用一个二维数组名赋值，如 p=str，使 p 指向二维数组 str 第 0 行的首地址。


注意　指针所指向的一维数组中的元素个数要与指针定义时的“常量表达式”相同。


小讲堂

2．指向一维数组的指针的运算及应用

假设有如下语句：

```
int  (*p)[5];
int   b[3][5];
p=b;
```

那么 p+1 指向什么地址空间？

（1）语句 int (*p)[5];声明了一个指向有 5 个整数元素的一维数组的指针。b 是一个 3*5 的二维整型数组，将数组名 b 赋值给指针 p，使指针 p 指向二维数组 b 的第 0 行。

（2）算术运算：p 是一个指向有 5 个整型元素的一维数组的指针，因此这个 1 的单位应该是 5*sizeof(int)=10 个字节，即使 p 指向数组 b 的第 1 行。

（3）将一个数组指针指向一个二维数组，访问二维数组中的每一行，即指针是一个指向一维数组的指针，因此，要保证二维数组中任一行中的元素个数应与定义时声明的指针所指的一维数组元素个数相同。

二、指针数组

【思考题 7-7】 数组是一组具有相同数据类型的数据的有序集合。指针也是一种数据，可不可以也作为数组中的元素呢，形成一个数据元素类型为指针的数组呢？



程序分析

数组定义的格式如下：

```
类型标识符    数组名[常量表达式];
```

其中，“类型标识符”用来说明数组中所有元素的数据类型，可以是 C 语言中任一种合法的数据类型。因为，只需要将“类型标识符”声明成指针类型，就可以定义一个数据类型为指针的指针数组。

请读者阅读下面的程序，并分析运行结果。



编写程序代码

```
#include  "stdio.h"
main()
{  int  x[]={1,2,3,4,5,6,7,8,9,10};
   int  i;
   int  *px[5];
   for(i=0;i<5;i++)
     px[i]=x+2*i;
    printf("%d,%d,%d,%d\n",*(*(px+2)+1),**px+1,*(px[3]),*(px[4]+1));
}
```



调试运行程序

程序运行结果：6,2,7,10



3．指针数组

指针数组就是数组元素为指针的数组。

（1）指针数组的定义格式如下：

```
类型标识符    *指针数组名[常量表达式];
```

如语句 int *px[5];定义了一个可以含有 5 个整型指针的数组 px，即 px 中的每个元素都是指向整型数据类型的指针。

（2）初始化。指针数组可以在定义的同时被初始化，一般用来保存指向字符串的指针。如语句：

```
char *lesson[3]={"data structure","computer design", "c language"};
```

lesson 是一个指针数组，它在定义的同时初始化。指针数组 lesson 有 3 个元素，每个元素都是一个字符指针，存放的是字符串的地址。其中 lesson[0]指向字符串"data structure"，lesson[1]指向字符串"computer design"，lesson[2]指向字符串"c language"。lesson 数组中的元素如图 7-5 所示。

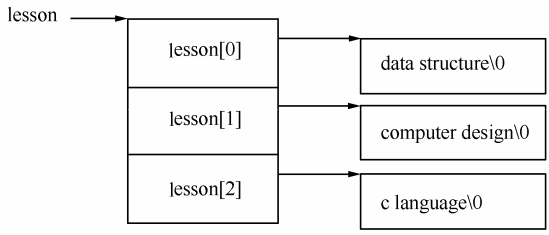


图 7-5　lesson 数组中的元素

（3）赋值。指针数组的赋值与普通数组相同，但要注意所赋予的值应该是地址值，而且所赋地址的类型要与指针数组的类型相同。



4．指向一维数组的指针与指针数组的区别

在学习过指向一维数组的指针和指针数组的定义、应用后，再来分析一下它们的区别。这是两个完全不同的概念，前者是指针，后者是数组。

（1）定义格式：仅差一对圆括号。

① 指向一维数组的指针。假设有定义语句：

```
int (*p)[4];
```

由于括号的优先级最高，因此 p 先与“*”结合，这表明 p 是指针变量，（*p）后面的[4]表明指针变量 p 指向一个一维数组，这个数组有 4 个元素，（*p）前面的 int 表明数组元素的类型为整型。

② 指针数组。假设有定义语句：

```
int *p[4];
```

由于方括号比“*”优先级高，因此 p 先与[4]结合，形成 p[4]，这表明 p 是一维数组，它有 4 个元素。而 p 前面的 int *表明 p 是指针类型的数组，每个元素都是一个整型指针。

（2）应用场合：前者是一个指针，后者是多个指针。

① 指向一维数组的指针。一般将一个二维数组名赋值给一个指向一维数组的指针，之后就可以像使用二维数组名那样使用指针，从而来访问二维数组中的某一行。

② 指针数组。适合存放若干个字符串，使字符串的处理更加方便灵活。对于多个字符串的存取，可以定义一个二维字符数组，也可以定义一个字符指针数组，事实上字符指针数组是用来存取多个字符串的一种最好的办法。

例如：

```
char *menu1[]={"Copy","Cut","Paste","Delete"};
char *menu2[4][7]={"Copy","Cut","Paste","Delete"};
```

其中，menu1 是字符指针数组，它有 5 个元素，每个元素都指向一个字符串，字符串的长度可以任意。menu2 是二维字符数组，可以存放 5 个字符串，每个字符串的最大长度是 7。

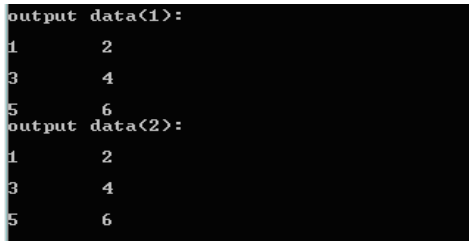
如果二维字符数组中存放的字符串长度各不相同，其中一个字符串的长度比较大，则要求二维字符数组的列数按此最大长度定义，这样就会浪费部分没有占用的内存单元。而用字符指针数组处理字符串不仅可以节省内存，还可以提高运行效率，请读者多体会。

练一练

【练习 7-6】 分析下面程序的运行结果，使用指向一维数组的指针访问二维数组元素。

```
#include "stdio.h"
main()
{  int  i,j,*p;
   int  (*pa)[2];
   int  a[3][2]={1,2,3,4,5,6};
   pa=a;
   printf("\noutput data(1): ");
   for(i=0;i<3;i++)
   {  printf("\n\n");
      for(j=0;j<2;j++)
        printf("%d\t",*(*(pa+i)+j));
   }
    printf("\noutput data(2): ");
    p=a[0];
    for(i=0;i<3;i++)
  {  printf("\n\n");
     for(j=0;j<2;j++)
       printf("%d\t",*(p++));
  }
    printf("\n");
}
```

程序运行结果如下：



解：程序分析：

（1）数组名 a 和指针 pa 都是行指针，指针 p 是列指针，通过 pa=a 和 p=a[0]分别给指针 pa 和 p 赋值。指针 pa 和 p 都可以实现将二维数组 a 中的元素输出。

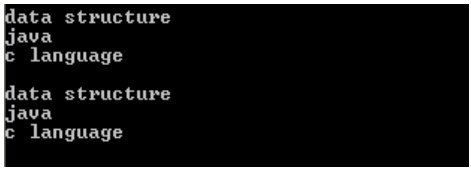
（2）a 是指针常量，pa 和 p 是指针变量，表达式 pa++使指针 pa 指向下一行，表达式 p++使指针 p 指向下一个元素。

【练习 7-7】 分析下面程序的运行结果，学习指针数组的使用。

```
#include "stdio.h"
#include "string.h"
main()
{  char  *lesson[3]={"data structure","java", "c language"};
   int  i;
   char  *cp;
   for(i=0;i<3;i++)
     printf("\n%s",lesson[i]);
   printf("\n");
   for(i=0;i<3;i++)
```

```
        {  cp=lesson[i];
        printf("\n%s",cp);
        }
    }
```

程序运行结果如下：



解：该程序的功能如下：

首先定义了一个指向 3 个字符串的字符指针数组 lesson，然后使用一个循环语句，将这 3 个字符串输出。最后使用指向字符数组的指针 cp 指向 lesson 字符数组的第 0 个元素，然后使用指针 cp 来输出这 3 个字符串。

想一想

学习本讲后，请读者思考关于指针与数组的关系、应用如何推广到多维数组呢？

本讲小结

本讲主要学习的是指向一维数组的指针与指针数组，通过学习要求能够掌握：

（1）使用指向一维数组的指针访问数组元素；

（2）使用指针数组访问多个字符串。

第十八讲　指针与函数

学习目标

- 学习指针作为函数参数，实现“地址传递”；
- 了解指针与函数的关系，应用指针函数与函数指针；
- 带参数的 main()函数及其应用。

一个变量有地址，一个数组有地址，一个函数也有地址。指针可以指向变量，也可以指向数组，还可以指向函数。

指针与函数的关系，主要表现在三个方面：指针作为函数参数、函数的返回值是指针（指针函数）和指向函数的指针（函数指针）。

一、指针作为函数参数

【思考题 7-8】　输入两个整数 a、b，要求用函数来按先大后小的顺序输出 a 和 b。

程序分析

因为函数的参数可以是普通变量，也可以是指针，所以我们将两个数的地址作为函数参数，传到函数中。在函数里面交换两个指针所指变量的值，即相当于交换了主函数中的两个变量的值。注意以下程序中函数参数的定义及使用方法。

请读者先阅读下面的程序代码。

编写程序代码

```
#include  "stdio.h"
void  swap(int  *p1,int  *p2)
{  int  temp;
   temp=*p1;      /* 将指针 p1 中的数据保存到临时变量 temp 中 */
   *p1=*p2;       /* 将指针 p2 中的数据保存到 p1 所指向的空间 */
   *p2=temp;      /* 将 temp 中的指针 p1 中的数据保存到 p2 所指向的空间*/
}
main()
{  int  a,b,*pa,*pb;
   printf("input a  b= ");
   scanf("%d %d",&a,&b);
   printf("  a=%d   b=%d\n",a,b);
   pa=&a;
   pb=&b;
   if(a<b)  swap(pa,pb);  /* 数据交换，使 pa 中保存大数的地址 */
   printf("  a=%d   b=%d\n",a,b);
   printf("*pa=%d *pb=%d\n",*pa,*pb);
}
```

调试运行程序

程序运行结果如下：

```
input a  b= 5  8
  a=5   b=8
  a=8   b=5
*pa=8 *pb=5
```

小讲堂

1．指针变量的“地址传递”：在子函数中交换指针变量所指的值

（1）【思考题 7-8】中，若执行到语句：

```
if(a<b)  swap(pa,pb);
```

指针变量 pa 和 pb 是作为实际参数传递给形式参数 p1 和 p2 的，分别传递的是主程序中变量 a 和 b 的地址。

（2）子函数中语句：

```
temp=*p1;
*p1=*p2;
*p2=temp;
```

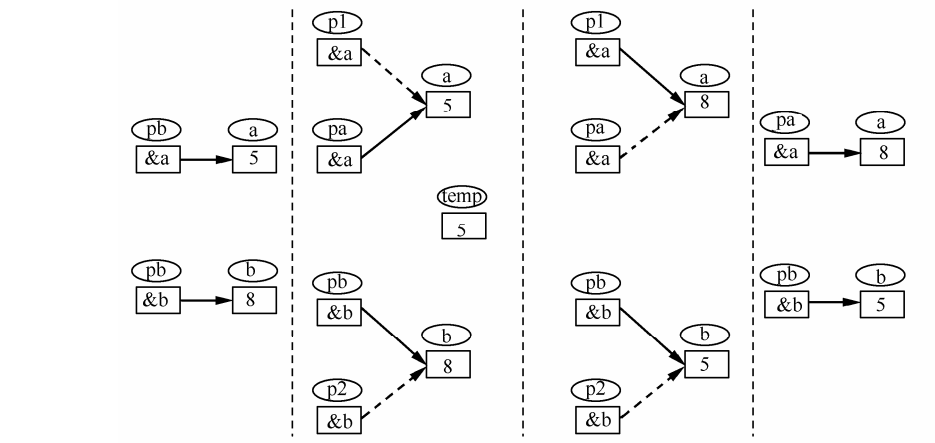
实现了将指针 p1 和指针 p2 指向的数据进行交换，实际上是交换了主程序中 a 和 b 的值，指针变量 p1 和 p2 的值并没有改变。

（3）swap 函数返回时，已经修改了实参指针变量 p1 和 p2 的所指变量的值，因此主程序中的语句：

```
printf("  a=%d   b=%d\n",a,b);
```

执行结果是 a=8 b=5，即比较大小后大数保存在变量 a 中，小数保存在变量 b 中。

（4）程序执行过程如图 7-6 所示。



调用 swap 函数前　　swap 函数执行前　　swap 函数返回前　　swap 函数返回后

图 7-6　数据交换前后示意图



2．指针变量的“地址传递”：在子函数中交换指针变量的值

阅读并分析下面程序的运行结果，注意比较子函数与【思考题 7-8】的区别。

```
#include "stdio.h"
void swap(int *p1,int *p2)
{ int *p;
  p=p1;     /* 指针值交换 */
  p1=p2;
  p2=p;
  printf("*p1=%d *p2=%d\n",*p1,*p2);
}
main()
{ int a,b,*pa,*pb;
  printf("input a  b= ");
  scanf("%d %d",&a,&b);
  printf("  a=%d   b=%d\n",a,b);
  pa=&a;
  pb=&b;
  if(a<b) swap(pa,pb);
  printf("  a=%d   b=%d\n",a,b);
  printf("*pa=%d *pb=%d\n",*pa,*pb);
}
```

程序运行结果如下：

```
input a  b= 5  8
  a=5    b=8
*p1=8 *p2=5
  a=5    b=8
*pa=5 *pb=8
```

程序分析如下：

（1） 执行语句 if(a<b) swap(pa,pb);调用子函数时，也是将指针变量 pa 和 pb 作为实际参数传递给形式参数 p1 和 p2 的，分别传递的是主程序中变量 a 和 b 的地址。

（2）子函数中语句：

```
p=p1;
p1=p2;
p2=p;
```

实现的是指针变量 p1 和 p2 的值进行交换，交换后 p1 指向的是变量 b，而 p2 指向的是变量 a。因此*p1=8，*p2=5。

（3）swap 函数返回时，p1 和 p2 的值并不能传递给实参 pa 和 pb。因为 C 语言中实参变量和形参变量之间的数据传递是单向的“值传递”方式。指针变量作函数参数也要遵循这一规则。子函数中不可能改变实参指针变量的值，但可以改变实参指针变量所指变量的值。

（4）程序执行过程如图 7-7 所示。

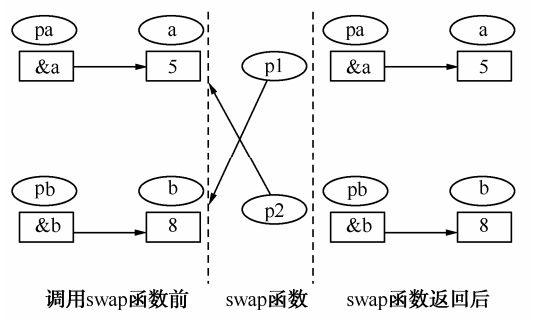


图 7-7　指针变量“值传递”前后示意图

二、指针函数

【思考题 7-9】　设计一个函数，在一个字符串中查找一个指定的字符。如果指定字符在字符串中，返回串中指定字符出现的第一个地址；否则，返回 NULL 值。要求使用指针函数实现在一个字符串中查找一个指定的字符。

程序分析

首先分析这个函数要实现的功能，就是在一个字符串中查找指定的字符第一次出现的位置。函数定义应包括函数的返回类型、函数名、形参表和函数体，我们按照这个思路确定。

（1）按照功能要求，函数应该提供两个形式参数，一个是待查找的字符串，一个是指定的用于查找的字符。

（2）函数的返回值应该是一个地址值，对应的是找到的字符第一次出现的地址，也就是指向这个位置的指针。如何在返回值类型中体现出返回值是指针类型呢？

（3）函数名可定义为 myStrCh。

（4）为了调试这个函数，需要编写 main()函数，在 main()函数中调用函数 myStrCh，函数的实际参数是待查找的字符串和用于查找的指定字符，并通过 printf 语句输出函数的返回结果，即指针值。

编写程序代码

```
#include "stdio.h"
char *myStrCh(char *str,char ch);
main()
{ char *p,ch,line[20];
```

```
      printf("\ninput str: ");
      gets(line);
      printf("\ninput ch: ");
      ch=getchar();
      p=myStrCh(line,ch);
      if(p!=NULL)
      {  printf("\nstring start at address: %x\n",line);
         printf("\nfirst occurrence of char ' %c ' is address: %x\n",ch,p);
         printf("\nThis is position of %d (start from 0)\n\n",p-line);
   }
}
char  *myStrCh(char  *str,char  ch)
{  while(*str!='\0')
   if(*str==ch)  return(str);
   else       str++;
  return NULL;
}
```



调试运行程序

程序运行结果如下：

```
input str: afg
input ch: g
string start at address: ffc2
first occurrence of char ' g ' is address: ffc4
This is position of 2 (start from 0)
```



3．指针函数

（1）函数可以返回一个整型值、实型值、字符型值，也可以返回一个指针值。指针函数就是指返回指针值的函数。返回指针值的函数原型如下：

```
基类型   *函数名(形参表);
```

其中，“基类型”是函数返回的指针所指向数据的类型。

（2）【思考题 7-9】的 main 函数中，line 是字符数组名，指向从屏幕输入的字符串"afg"；ch 是字符变量，存放待查找的指定字符'g'。函数调用 myStrCh(line,ch)返回的就是字符'g'在字符串"afg"中第一次出现时的地址，并赋给指针变量 p。line 是字符串"afg"中首字符'a'的地址，p 是字符串中字符'g'第一次出现的地址，p-line 就是第一个字符'g'在字符串中的序号。注意：这个序号是从 0 开始计算的。

三、指向函数的指针

【思考题 7-10】 设计一个程序，通过指向函数的指针调用所指向的函数。



程序分析

C 语言中的指针，不仅可以指向整型、字符型和实型等变量，还可以指向函数。我们可

以定义一个指针变量，使它的值等于函数的入口地址，即指针是指向这个函数的，通过这个指针变量就可以调用此函数。

请读者阅读下面的程序，来学习指向函数的指针。



编写程序代码

```
#include "stdio.h"
int add(int x, int y);
main()
{ int x,y;
   int total1,total2;
   int (*pt)(int x, int y);
   printf("\ninput x and y: ");
   scanf("%d", &x);
   scanf("%d", &y);
   pt=add;
   total1=add(x,y);
   total2=(*pt)(x,y);
   printf("\n(1) %d + %d = %d", x,y, total1);
   printf("\n(2) %d + %d = %d", x,y, total2);
   printf("\n");
}
int  add(int x, int y)
{ return(x+y); }
```



调试运行程序

程序运行结果如下：

```
input x and y: 5 6

(1) 5 + 6 = 11
(2) 5 + 6 = 11
```



4．指向函数的指针

C 语言中的每一个函数经过编译后，其目标代码在内存中连续存放，该代码的首地址就是函数执行时的入口地址。在 C 语言中，函数名本身就代表着该函数的入口地址。通过这个入口地址可以找到该函数，该入口地址称为函数的指针。

C 语言中可以定义一个指针变量，使它的值等于函数的入口地址，即指针是指向这个函数的，通过这个指针变量就可以调用此函数。这个指针变量称为指向函数的指针，又称为函数指针。

（1）函数指针的定义格式。因为这个指针是指向函数的，所以它的定义是与某个函数原型相关的。如，有一个函数的原型为：返回类型　函数名(形参表);则可用于指向这个函数的指针变量应该定义为：

```
返回类型  (*指针变量名) (形参表);
```

在【思考题 7-10】中，函数的原型为：int　　add(int x,int y);则对应的函数指针的定义：

```
int (*fp)(int x,int y);
```

可见，将函数原型中的函数名替换为(*变量名)，就可以定义一个指向该函数的指针变量。

（2）函数指针的赋值。定义了函数指针之后，首先必须将一个函数名（代表该函数的入口地址）赋给函数指针，然后才能通过函数指针间接调用这个函数。指针的赋值格式为：

```
函数指针变量名=函数名;
```

在【思考题 7-10】中，语句：pt=add;实现了函数指针的赋值。

（3）函数调用：可以有两种方法来调用函数。

① 通过函数名直接调用函数。调用格式如下：

```
函数名(实参表);
```

如【思考题 7-10】中的语句：total1=add(x,y);

② 通过函数指针间接调用函数。调用格式如下：

```
(*变量名)(实参表);
```

如【思考题 7-10】中的语句：

```
total2=(*pt)(x,y);
```



注意 在利用函数指针来间接调用其所指向的函数时，该函数的定义必须存在，否则将出现错误。



5．指针函数与函数指针的区别

这是两个完全不同的概念，前者是函数，后者是指针。

（1）定义格式：仅差一对圆括号。

① 指针函数。假设有定义语句：

```
int  *fp(int x,int y);
```

由于括号的优先级最高，fp 先与其后的(int x, int y)结合，表明 fp 是一个函数，该函数有两个整型参数。fp 前面的 int *表明函数 fp 的返回值是 int 类型的指针。因此 fp 是一个指针函数。

② 指向函数的指针（函数指针）。假设有定义语句：

```
int  (*fp)(int x,int y);
```

由于括号的优先级最高，而且从左向右结合，因此 fp 先与“*”结合，这表明 fp 是指针变量，(*fp)后面的(int x,int y)表明指针变量 fp 指向一个函数，所指的函数有两个整型参数，(*fp)前面的 int 表明指针 fp 所指函数的返回值是 int 类型。因此 fp 是一个指向函数的指针变量。

（2）应用场合：前者是一个函数，只是函数的返回值是指针；后者是一个指向函数的指针，通过这个指针可以间接地调用所指向的函数。

① 指针函数。非指针型函数调用后，可以返回一个变量，即每次调用只能返回一个数

据，有时需要从被调用函数返回一批数据到主调函数中，这时可以通过指针函数来解决，返回一批顺序存储的数据，多应用在数组或字符串的一些操作中。在 C 语言中，很多库函数都是通过指针函数实现的，用途非常广泛。

② 指向函数的指针（函数指针）。一个函数指针可以先后指向不同的函数，因此利用这种方式可以选择合适的被调用函数，而且，还可以将函数指针作为函数的参数来应用。

四、带参数的main函数及其应用

【思考题 7-11】 我们之前学习中定义的 main 函数，其参数表都是空的，那么 main 函数可以带参数吗？程序运行时参数又如何传递给 main 函数呢？编写一个程序，显示带参数的命令行中的命令名和参数。

程序分析

一个最简单的 C 语言源程序可以由头文件和 main 函数组成，main 函数是程序执行的入口。main 函数也是一个函数，因此在进行函数定义时，也是可以定义带参数的 main 函数的。当程序运行自动调用带参数的 main 函数时，在命令行中输入实际参数，程序才能继续执行。

我们通过一个简单的程序来学习带参数的 main 函数的定义及应用。

编写程序代码

```
#include  "stdio.h"
main(int argc, char *argv[])
{  int i;
   printf("\n The information of the command line with parameter are:\n");
   for(i=0;i<argc;i++)  printf("argv[%d]:%s\n",i,argv[i]);
}
```

调试运行程序

（1）执行程序。假设【思考题 7-11】经编译、连接后生成的可执行文件为 21_4.exe，存于 c:\tc 目录下，则在 DOS 提示符下输入如下命令运行程序：

```
c:\tc>21_4   happy  new  year↙
```

注意 这里是执行 Windows 系统的命令提示符，进入该文件所在目录（c:\tc）后再输入上行命令才能得到如下结果，在 TC 下的 Run 命令不能得到如下结果（因为不允许输入参数值）。

（2）程序运行结果如下：

```
C:\tc>21_4 happy new year

The information of the command line with parameter are:
argv[0]: C:\TC\21_4.EXE
argv[1]: happy
argv[2]: new
argv[3]: year
```

小讲堂

6．带参数的main函数及其应用

由不带参数的 main 函数所生成的可执行文件，在执行时只能输入可执行文件名（即命令名），而不能输入参数；而在实际应用中，经常希望执行程序时，能够由带参数的命令行向程序提供所需要的信息，这就需要在程序中定义带参数的 main 函数。

（1）命令行参数。所谓“命令行”，是指在 DOS 提示符下输入的命令名（.exe 文件）及其参数，其中的参数称为命令行参数。带参数的命令行一般具有如下格式：

```
命令名  参数 1    参数 2   …   参数 n
```

命令名和参数以及参数和参数之间都由空格隔开，这些参数是通过命令行传递到程序中。

（2）带参数的 main 函数。main 函数通常是不带参数的，若带参数可以带两个参数，函数的参数表格式如下：

```
main(int   argc,char   *argv[]);
```

第一个形参 agrc 是一个整型变量，存放的是命令行中命令名与参数的总个数。第二个形参 agrv 是一个指针数组，其元素可以指向带参数的命令行中命令名与参数所代表的字符串。

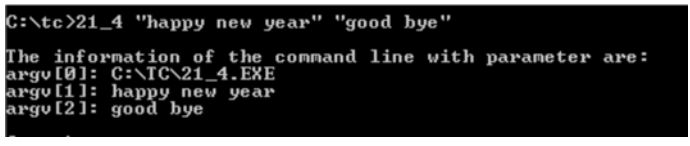
（3）程序分析。【思考题 7-11】中 main 函数的参数 agrc 和 argv 的值如下所示：

```
argc=4;
argv[0]指向被运行程序的全路径程序文件名;
argv[1]指向字符串"happy"
argv[2]指向字符串"new"
argv[3]指向字符串"year"
```

注意 若某个参数中是含有空格的，则需要将这个参数用双引号括起来。例如，执行上例时，若在 DOS 提示符下输入：

```
c:\tc\21_4   "happy  new year"   "good bye"
```

则屏幕显示如下：



练一练

【练习 7-8】 分析下面程序的运行结果，学习指针函数的应用。

```
#include  "stdio.h"
char  *mystrcat(char  *str1,char  *str2);
main()
{  char  str1[40];
   char  str2[20];
   char   *p;
```

```
    printf("\ninput str1: ");
    gets(str1);
    printf("\ninput str2: ");gets(str2);
    p=mystrcat(str1,str2);
    printf("\noutput str1    : %s\n",str1);
    printf("\noutput str2    : %s\n",str2);
    printf("\noutput str1 new : %s\n",p);
}
char  *mystrcat(char  *str1,char  *str2)
{  char  *str=str1;
   while(*str)   str++;
   while(*str2)  *str++=*str2++;
   *str='\0';
   return(str1);
}
```

解：程序分析如下：

（1）该程序功能是将字符串 str2 追加到 str1 后，即模拟实现标准库函数 strcat()。

（2）参数 str1 必须有足够的空间能够容纳连接后的字符串，即 str1 能够容纳的实际字符个数是：strlen(str1)+strlen(str2)+1。

程序运行结果如下（假设 str1="hello"，str2="world"）：

```
input str1: hello
input str2: world
output str1     : helloworld
output str2     : world
output str1 new : helloworld
```

【练习 7-9】 已知契比雪夫多项式的定义如下所示：

x	(n=1)
2*x*x−1	(n=2)
4*x*x*x−3*x	(n=3)
8*x*x*x*x-8*x*x+1	(n=4)

设计一个程序，从键盘输入整数 n 和浮点数 x，计算多项式的值。

解：源程序如下：

```
#include  "stdio.h"
char  *mystrcat(char  *str1,char  *str2);
main()
{  float  fn1(float x),fn2(float x),fn3(float x),fn4(float x);
   float  (*fp)(float x);
   float  x;int   n;
   printf("\ninput x: ");
   scanf("%f",&x);
   printf("\ninput n: ");
   scanf("%d",&n);
   switch(n)
   {  case 1:  fp=fn1;   break;
      case 2:  fp=fn2;   break;
      case 3:  fp=fn3;   break;
      case 4:  fp=fn4;   break;
```

```
        default: printf("data input error!");
        return;
      }
     printf("result=%f",(*fp)(x));
   }
   float  fn1(float  x)
   {return(x);  }
   float  fn2(float  x)
   {return(2*x*x-1);    }
   float  fn3(float  x)
   {return(4*x*x*x-3*x);}
   float  fn4(float  x)
   {return(8*x*x*x*x-8*x*x+1);  }
```

程序运行结果如下：



本讲小结

本讲主要学习的是指向函数的指针与指针函数，通过学习要求能够掌握：

（1）指针作为函数参数；

（2）使用指针函数返回一个地址值；

（3）使用指向函数的指针间接调用函数；

（4）带参数的 main()函数及其应用。

想一想

指针函数与函数指针的应用有什么不同？在程序中使用函数指针，会提高程序的可读性吗？会提高程序的运行效率吗？

本章小结

本章着重介绍了各种类型的指针及其定义、指针的基本运算、指针与数组的关系、指针与函数的关系，还介绍了带参数的 main 函数的定义及应用。

指针是专门用来存放地址的变量。指针除了可以指向基本数据类型变量的地址，还可以指向数组、函数。一个数组名就是一个指针常量，一维数组的数组名指向其首地址，二维数组的数组名指向其首行。数组作为参数传递时，实际传递的是可以访问该数组的指针。

本章介绍指向数组的指针时，还详细介绍了指针数组，特别是字符指针数组。带参数的 main 函数就是字符指针数组的典型应用。

函数名是指向该函数的指针常量，指向一个函数并与函数名类型相同的任何指针都可以像函数名那样调用该函数。函数作为参数传递时，实际传递的是调用该函数的指针。

课后习题七

一、选择题

1．已知：int a[]={1,3,5,7,9}，*p=a;，则值为 7 的表达式是_____。

A）*p+3　　B）*p+4　　C）*(p+3)　　D）*(p+4)

2．以下程序的输出结果是_____。

```
main()
{  int  a[10]={9,8,7,6,5,4,3,2,1,0},*p=a+2;
   printf("%d",*--p);
}
```

A）a[2]的地址　　B）7　　C）8　　D）9

3．在以下定义中，对标识符 ptr 描述正确的是_____。

```
int (*ptr)[3];
```

A）定义不合法

B）是一个指针数组名，每个元素是一个指向整型变量的指针

C）是一个指针，它指向一个具有 3 个元素的一维数组

D）是一个指向整型变量的指针

4．在以下定义中，不能给 a 数组输入字符串的语句是_____。

```
char  a[10],*b=a;
```

A）gets(a);　　B）gets(a[0]);　　C）gets(&a[0]);　　D）gets(b);

5．设 int a[]={1,2,3},i,*p=a;，根据定义及语法分析不正确的是_____。

A）a[p-a]　　B） *(&a[i])　　C） p[i]　　D）*(*(a+i))

6．若有如下的程序段：

```
char  str[]="hello";
char  *ptr=str;
```

则*(ptr+5)的值为_____。

A）'o'　　B）'\0'　　C）不确定的值　　D）'o'的地址

7．以下程序的输出结果是_____。

```
main()
{  int a=25, *p;
   p=&a;
   printf("%d", ++*p);
}
```

A）23　　B）24　　C）25　　D）26

8．若有如下定义: int a[2][3]={2,4,6,8,10,12}; 则如下描述不正确的是_____。

A）*(a+1)为元素 a[1][0]的指针　　B）a[1]+1 为元素 a[1][1]的指针

C）*(a+1)+2 为元素 a[1][2]的指针　　D）*a[1]+2 的值是 12

9．对于基类型相同的两个指针变量之间，不能进行的计算是_____。

A）<　　B） =　　C）+　　D）-

10．以下程序的输出结果是_____。

```
main()
{  int  a[]={2,4,6,8,10},y=1,x,*p;
   p=&a[1];
```

```
    for(x=0;x<3;x++)      y+=*(p+x);
    printf("%d", y);
}
```

A）17　　B）18　　C）19　　D）20

11．执行以下程序段后，m 的值是_____。

```
int  a[2][3]={{1,2,3},{4,5,6}};
int  m,*p;
p=&a[0][0];
m=(*p)*(*(p+2))*(*(p+4));
printf("%d", m);
```

A）6　　B）18　　C）15　　D）20

12．以下程序的输出结果是_____。

```
#include  "string.h"
main()
{  char  str[][20]={"hello","beijing"},*p=str[0];
   printf("%d",strlen(p+20));
}
```

A）0　　B）5　　C）7　　D）20

13．库函数 strcpy 用以复制字符串。如有以下定义和语句：

```
char str1[]="string", str2[8], *str3, *str4="string";
```

则对库函数 strcpy 的调用不正确的是_____。

A）strcpy(str1, "HELLO1");　　B）strcpy(str2, "HELLO2");

C）strcpy(str3, "HELLO3");　　D）strcpy(str4, "HELLO4");

14．以下程序的输出结果是_____。

```
main()
{  char  *p[5]={"ABCD","EF","GHI","JKL","MNOP"};
   char  **q=p;
   int  i;
   for(i=0;i<=4;i++) printf("%s",q[i]);
}
```

A）ABCDEFGHIJKL　　B）ABCD

C）ABCDEFGHIJKLMNOP;　　D）AEJM

15．若有语句 int　(*p)(); 则其中的标识符 p 是_____。

A）指向整型变量的指针　　B）指向整型函数的指针

C）返回整型指针的函数　　D）返回整型变量的函数

16．以下程序的输出结果是_____。

```
main()
{  char  *s="abcde";
   s+=2;
   printf("%s",s);
}
```

A）cde　　B）字符 c

C）字符 c 的地址　　D）无确定的输出结果

17．main()函数中参数表示形式不合法的是_____。

A）main(int a, char *c[]);　　B）main(int arc, char **arv);

C）main(int argc, char *argv);　　D）main(int argv, char *argc[]);

18．以下程序的输出结果是_____。

```
#include "string.h"
main()
{ char arr[2][4];
  strcpy(arr[0],"you");
  strcpy(arr[1],"me");
  arr[0][3]='&';puts(arr[0]);
}
```

A）you&me　　B）you　　C）me　　D）err

19．以下程序段正确的是_____。

A）int *p,x=6; *p=x;

B）int *s,k; *s=100;

C）int *s,k; char *p,c; s=&k; p=&c; *p='a';

D）int *s,k; char *p,c; s=&k; p=&c; s=p; *s=1;

20．以下程序的输出结果是_____。

```
void func(int *a,int b[]) { b[0]=*a+6;}
main()
{ int a, b[5];
  a=0;
  b[0]=3;
  func(&a,b);
  printf("%d",b[0]);
}
```

A）6　　B）7　　C）8　　D）9

二、填空题

1．以下程序的输出结果是 _____。

```
main()
{ int a[5]={2,4,6,8,10},*p,**k;
   p=a;
   k=&p;
   printf("%d  ",*(p++));
   printf("%d  ",**k);
}
```

2．以下程序的输出结果是 _____。

```
main()
{ int a[3]={2,4,6},*ptr=&a[0],x=8,y,z;
  for(y=0;y<3;y++) z=(*(ptr+y)<x)? *(ptr+y): x;
  printf("%d  ",z);
}
```

3．以下程序的输出结果是 _____。

```
main()
{ char b[]="ABCDEFG ",*chp=&b[7];
   while(--chp>=&b[0]) printf("%c  ",*chp);
}
```

4．以下程序的输出结果是 _____。

```
main()
```

```
{  char  s[20]="goodgood",*sp=s;
   sp=sp+2;
   sp="to";
   printf("%s",s);
}
```

5. 下面程序的输出结果是 _____。

```
main()
{  int  a[10]={6,7,2,9,1,10,5,8,4,3},*p,*s;
   for(p=a,s=a; p-a<10; p++)
     if(*p>*s)   s=p;
   printf("%d",*s);
}
```

6. 下面程序的输出结果是_____。

```
void  f(int i)
{  static  char  s[]="T&W";
   char  *p=s+i;
   while(--p>=s)  putchar(*p);
}
main()
{  f(1);
   f(3);
}
```

7. 下面程序的运行结果是_____。

```
main()
{  int  a=0,b=0;
   char  s[]="ahspks.ahedu.gov.cn",*t;
   t=s;
   while(*t)
   {  switch(*t++)
   {  case 'a':
      case 'h':
      default: a++;
      case '.': b++;
   }
 }
 printf("%d  %d\n",a,b);
}
```

8. 下面程序的运行结果是_____。

```
void  f(char  *p)
{  if(*p!=NULL)
   {  f(p+1);
      printf("%c",*p);
   }
}
main()
{  char  s[]="139";
   f(s);
}
```

三、编程题

1. 有一个整型二维数组，大小为 m 行 n 列，要求分别找出其中最大值所在的行和列，以及该矩阵中的

最大值。请编写一个函数 MaxVal，m 和 n 是该函数的形参，所求二维数组最大值通过函数值返回。数组元素的值在 main 函数中输入。

2．设计一个程序，实现运动员成绩管理。程序中有运动员姓名数组、运动员编号数组、运动员成绩数组，自定义数组名称及长度，并赋初值，实现按照运动员成绩从高到低打印出运动员信息。

3．假定有 5 个学生，每个学生有 4 门功课的成绩。设计一个程序，查找某学生的成绩。假设被查学生的序号为 0～4，要求以指向数组的指针作为函数形参。

第八章　结构体、共用体和枚举

在第四章中我们介绍了第一种构造类型——数组，数组中的各元素必须属于同一个数据类型。但是在实际生活中，我们经常会遇到有若干各不同类型的数据组合成一个有机的整体。例如一个学生的信息包括：学号、姓名、性别、年龄、成绩等，这些信息的类型各不相同，姓名为字符型数组，学号为整型，性别为字符型，成绩为浮点型等。为了解决这个问题，C 语言提供了另一种构造数据类型——结构体，它可以将不同类型的数据存放在一起。

C 语言还提供了一种构造数据类型——共用体，它可将不同类型的数据在不同时刻存放在同一个存储单元中。此外，C 语言还提供了枚举类型以及使用 typedef 类型定义来为原来已有的类型重新定义一个别名。

第十九讲　结构体基础



学习目标

- 掌握结构体类型和结构体类型变量的定义;
- 掌握结构体变量的使用（包括输入、输出和成员引用）;
- 掌握结构体类型的嵌套定义。

【思考题 8-1】　请将一个学生的学号，姓名，性别，语文、数学、英语成绩输入后存在计算机中并显示在屏幕上。



程序分析

以前我们所学习的数据类型都是单一数据类型，如整型、浮点型、字符型等，而数组中的每个元素也必须是一种数据类型。而这里要求的学生的“学号”可以用一个整型变量存储，“姓名”则用字符型数组，“性别”用一个字符变量存储，“语文”、“数学”、“英语”可以用三个整型变量存储。但这只是一个学生的信息，如果想存储多个学生的信息就不能直接用这样的简单类型来定义各个变量了。

C 语言提供给我们一种复合的数据类型，叫做结构体。在这种类型中，每个成员可以是不同的简单类型或者是其他的复合数据类型，这就给我们存储像“学生信息”这样的复杂的数据提供了极大的方便。使用结构体类型的数据，可以直接定义出学生这样一个结构体类型，再定义这样一个变量，并为这个变量中的每个成员赋值，就可以实现如学生信息这样的复合数据结构了。下面我们通过一个例程来了解结构体类型及其变量的定义、结构体类型变量的使用方法。



编写程序代码

```
struct  student                    /*定义学生结构体类型*/
{  int  num;                       /*定义学号*/
   char  name[20];                 /*定义姓名*/
   char  sex;                      /*定义性别*/
```

```
    int  chinese,math,english;     /*定义语文、数学、英语成绩*/
};
main()
{  struct  student  stu;          /*定义结构体类型的变量*/
   printf("\nplease input information of the student:");
   printf("\nnumber,name,sex,chinese,math,english\n");
   scanf("%d",&stu.num);
   scanf("%s",stu.name);
   getchar();     /*本行的功能是接收输入在 sex（性别）之前的空格字符*/
   scanf("%c",&stu.sex);
   scanf("%d%d%d",&stu.chinese,&stu.math,&stu.english);
   printf("The student information:");
   printf("\nnum:%d",stu.num);
   printf("\nname:%s",stu.name);
   printf("\nsex:%c",stu.sex);
   printf("\nchinese:%d",stu.chinese);
   printf("\nmath:%d",stu.math);
   printf("\nenglish:%d",stu.english);
}
```



调试运行程序

程序要求从键盘上输入学生的学号，姓名，性别（其中 f 代表女，m 代表男），语文、数学、英语成绩，并将存入的结果显示在屏幕上。



注意 在这个程序中字符变量 ch 的作用就是为了接收在输入各值的时候，在输入 name 之后，sex 之前的那个空格；或是每输入一个值后回车时，用来接收回车符。如果没这个变量接收，程序会出错。

程序运行结果如下：

```
please input information of the student:
number,name,sex,chinese,math,english
1 Limei f 98 86 92
The student information:
num:1
name:Limei
sex:f
chinese:98
math:86
english:92_
```



1．结构体类型的定义

结构体是一种复合的数据类型，也就是将多种数据类型封装在一起，形成一个新的数据类型。其定义格式如下：

```
struct    结构体类型名
{    成员列表
};
```

其中，“struct”是结构体类型的关键字，不能省略，表示后面定义的类型是一个结构体类型，“结构体类型名”为合法的标识符。注意右花括号后面有一个分号“;”作为结构体类型定义的结束标志，不能省略。

花括号内的成员列表用来定义组成该结构体类型的各个成员，对每个成员应进行类型说明，方法与定义其他简单类型变量一致，其说明格式为：

```
类型符  成员名;
```

在本讲【思考题 8-1】的例程中，就定义了一个学生结构体类型。这个类型中包括学号，姓名，性别，语文、数学、英语三门成绩。定义如下：

```
struct  student                    /*定义学生结构体类型*/
{  int  num;                       /*定义学号*/
   char  name[20];                 /*定义姓名*/
   char  sex;                      /*定义性别*/
   int  chinese,math,english;      /*定义语文，数学，英语成绩*/
};
```

例如再定义一个出生时间（birthday）的结构体类型，包括年（year）、月（month）、日（day）三个成员，可以定义为三个整型变量。类型定义如下：

```
struct  birthday
{  int  year;                  /*出生年份*/
   int  month;                 /*出生月份*/
   int  day;                   /*出生日期*/
};
```

在定义了一个结构体类型时，应注意以下几个问题：

（1）在一个结构体类型的定义中，其成员类型可以是除本身结构体类型之外的任何已有类型。（如果是定义指针类型成员可以为自身类型，在后面链表中会有介绍。）

（2）当一个结构类型定义在函数之外时，它具有全局作用域（如本讲例题中的结构体类型就具有全局作用域），若定义在任一对花括号之内时，则具有局部作用域（即有效范围）。

（3）在程序中同一个作用域内结构体类型名是唯一的，即不允许出现重复的类型标识。

（4）每个结构体类型定义中的成员名在该类型中必须唯一，但在整个程序中不要求唯一（即不同的结构体类型内定义的成员之间可以相同）。

（5）类型定义语句属于非执行语句，只在程序编译阶段处理它，并不在编译后生成的目标程序中存在对应的可执行目标代码。



2．结构体变量的定义

有了结构体类型，它仅相当于一个模型，但其中并无具体数据，系统不为它分配实际内存单元。为了能在程序中使用结构体类型的数据，就须在此类型基础上定义变量。就像我们不能直接给 int 类型赋值为具体数字 3（例如 int=3;是不允许的）一样，必须在定义了这种类型的变量后，才能将具体的值赋给这个变量。结构体类型定义后，就可以用它来定义结构体类型变量了。结构体类型的变量定义可以有以下三种方式。

1）先声明结构体类型再定义结构体变量

这种方法是先定义出结构体类型，然后再使用这个类型来定义所需变量。

```
struct    结构体类型名
{   成员列表
};
struct    结构体类型名   变量名表;
```

例如，使用本讲【思考题 8-1】例程中的结构体类型 student 定义两个该类型的变量 s1 和 s2，语句如下：

```
struct  student
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
};
struct  student  s1,s2;          /*定义 s1、s2 为 student 结构体类型变量*/
```

这种定义方法就是先定义出结构体类型，然后使用 struct 和结构体类型名再加上多个变量的定义方法。注意关键字 struct 不能省略，这种方法是最通常的定义方法。在反复使用结构体类型来定义变量时，为了简化书写代码，也可以用#define 定义一个符号常量来代表一个结构体类型。例如：

```
#define  STUD  struct  student;
```

这时，就可以用 STUD 来代替 struct student 来定义变量：

```
STUD  s1,s2;
```

这个定义语句与语句 struct student s1,s2;等价。

2）在定义结构体类型的同时定义结构体变量

这种方法是在定义出结构体类型的同时直接定义所需变量，好处是可以简化语句。

```
struct  结构体类型名
{   成员列表
}变量名表;
```

例如上面的变量定义语句可改为：

```
struct  student
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
}s1,s2;      /*在定义结构体类型 student 时直接定义出变量 s1、s2*/
```

1）和 2）两种定义方法都可以在程序不同位置多次定义该类型的变量（例如在多个函数中都可以使用该类型来定义变量）。

3）直接定义结构体变量

可以省略结构体类型名来定义一个结构体类型。

```
struct
{    成员列表
}变量名表;
```

这种方法可以不指明结构体类型名而直接定义出各个变量，但有一个缺点，就是这种定义方法只能在定义类型的同时直接定义出变量，在以后程序其他位置因为没有定义结构体类型名而无法再定义其他该类型变量。

例如上面的定义还可改为：

```
struct
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
}s1,s2;                              /*定义结构体变量 s1 和 s2*/
```

结构体变量在定义后，系统就为该变量分配了一段连续的存储单元。例如：

```
struct  data
{  int  a;
   float  b;
   char  c;
}d1;
```

上述变量 d1 的内存分配如图 8-1 所示，结构体变量 d1 共占有 7 个连续内存单元，a、b、c 三个成员所占的字节数分别是 2、4、1。

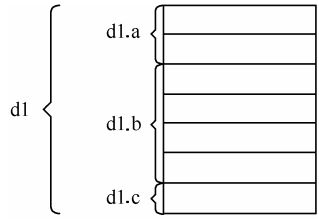


图 8-1　结构体变量 data1 内存分配示意图

结构体类型变量占内存大小是它里面各成员所占内存大小之和，可以用 sizeof 运算符测出一个结构类型或结构变量的字节数。例如，表达式 sizeof(data)或 sizeof(d1)的值都是 7。

定义了结构体型变量之后，结构体类型名的任务就完成了，在后续的程序中除求类型的长度和强制类型转换外，不再对其操作，而只对这些变量（如 s1、s2 等）进行赋值、存取或运算等操作。在编译时，对类型不分配空间，只对变量分配空间。结构体型变量所占内存空间是各成员变量所占内存单元的总和，各成员间占用的存储单元是连续的。

就好像指针变量一样，有些是指向整型数据的指针变量，有些是指向实型数据的指针变量，后面还会讲到指向结构体类型数据的指针变量。对于结构体类型变量也类似，有些是 struct student 的结构体变量，有些是 struct data 的结构体变量，有些是其他结构体类型的结构体变量。如果说指针变量要看它指向何种类型的指针，那么结构体变量要看它到底是哪种具体的结构体类型的变量。结构体是构造类型，不像基本类型在任何时候其含义是一致的，用户可以构造多个结构体类型，甚至对同一结构体标记在不同的程序中，其含义可以不同（指其成员不完全相同）。这一点一定要引起足够的重视。

小讲堂

3．结构体类型变量的初始化

同其他类型变量一样，在定义变量的同时，可以为其每个成员赋初值，称为结构体变量的初始化。所有结构体变量，不管是全局变量还是局部变量，自动变量还是静态变量均可如此。结构体变量的初始化的一般格式如下：

```
struct  结构体类型名  变量名={初值表};
```

其中，初值表为各成员的初值表达式。注意：初值表达式的类型应与对应成员的类型相同，否则会出错。各初值表达式之间用逗号分隔。例如：

```
struct  student
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
};
struct  student  s1={1,"Limei",'f',98,86,92},s2;  /*变量 s1 初始化*/
```

通过以上定义和初始化，结构体变量 s1 的各成员的初始值如下：成员 num 的值为 1，成员 name 的值为字符串"limei"，成员 sex 值为'f'，成员 chinese 的值为 98，成员 math 的值为 86，成员 english 的值为 92。

注意 定义不能改写为：

```
struct  student  s1,s2;
s1={1,"limei",'f',98,86,92};
```

这是因为 C 语言不允许将一组常量通过赋值运算符直接赋给一个结构体变量。同简单变量一样，当全局或静态结构体类型的变量未被初始化时，它的每个成员被系统自动置为 0；当自动结构类型的变量未被初始化时，它的每个成员的值是随意的，即不确定的。

小讲堂

4．结构体成员的引用

定义结构体变量之后就可以利用它存取具体结构体数据。系统为结构体变量提供的运算有赋值（=）、直接引用成员（.）和间接引用成员（->）三种，分别称为赋值运算符、直接成员运算符（点运算符）和间接成员运算符（又称箭头运算符）。其中成员运算符同下标运算符（[]）一样具有最高的优先级，而赋值运算符的优先级别较低。其中，间接引用成员（->）将在下一讲中详细讲解。在使用这些运算符的时候要注意以下几点。

（1）赋值运算符的两边应为同类型的结构体变量，即为同一结构体类型标识符所定义的变量。例如：

```
struct  student  s1={1,"Limei",'f',98,86,92}, s2;
s2=s1;     /*将 s1 各成员的值赋给 s2 的各成员*/
```

其中，表达式 s2=s1;是将结构体变量的各个成员的值分别赋给同类型结构体变量 s2 的相应成员，两个变量内各成员值相同。结果 s2 的各成员的值为：成员 num 的值为 1，成员 name 的

值为字符串"limei"，成员 sex 值为'f'，成员 chinese 的值为 98，成员 math 的值为 86，成员 english 的值为 92。

（2）直接引用结构体成员运算符的左边应该是一个结构体变量，右边应该是该结构体变量中的一个成员，运算结果是一个结构中的成员变量，不能不使用结构体变量名而直接使用结构体成员变量名来表示该成员。直接引用结构体变量成员的一般格式如下：

```
结构体变量名.成员名
```

例如上例中，s1 的 num 成员可以写为 s1.num，而不能直接写成 num 来表示它。



5．结构体类型变量的赋值

结构体变量是一个复合构造类型的变量，我们在赋值的时候只能对该结构体变量的每个最底层成员进行赋值，而不能直接对结构体变量本身赋值。如在程序中有语句：

```
scanf("%d",&s1.num);
```

就是对结构体变量 s1 的 num 成员进行赋值（从键盘输入数据）。

结构体类型变量可以整体引用来赋值。假设变量 s1 中各成员已赋有初值，可以有如下整体赋值语句：

```
s1=s2;
```

即将变量 s1 的所有成员的值一一赋给变量 s2 的各成员。

结构体型变量只能对逐个成员进行输入或输出，不可进行整体的输入或输出，如：

```
scanf("%d%d%d",&stu.chinese,&stu.math,&stu.english);
```

是对【思考题 8-1】的例程中变量 stu 的三个成员进行赋值，但如果写成如下语句是错误的：

```
scanf("%d%s%c%d%d%d",stu);
```



6．结构体类型的嵌套

结构体类型的定义可以进行嵌套，即一个结构体类型里的某一成员本身又是一个结构体类型。这时我们可以将里面的结构体先定义，外层的结构体类型后定义。如一个学生信息里要是想加上出生日期时，出生日期应该包括年、月、日三个值就需要先定义一个结构体类型“出生日期”，然后再定义结构体类型“学生”。定义语句如下：

```
struct  birthday                /*定义结构体变量 birthday */
{  int  year;                   /*出生年份*/
   int  month;                  /*出生月份*/
   int  day;                    /*出生日期*/
};
struct  student                 /*该结构体类型定义为学生信息*/
{  int  num;
   char  name[20];
   char  sex;
    struct  birthday  bir;      /*定义出生日期的成员*/
   int  chinese,math,english;
};
struct  student  s1,s2;
```

这时如果想表示变量 s1 的成员 bir 中的成员 year，就可以用两级直接引用符号来表示，如 s1.bir.year，语句 s1.bir.year=1998;即给成员 s1.bir.year 赋初值为 1998。

在 C 语言的运算符中，取成员运算符“.”优先级最高，若结构体定义是嵌套的，则只能引用最低级的成员（用若干“.”运算符，逐级引用到最低级）。如 s1.bir.year 是合法的，而 s1.bir 或 s1.year 都是非法的。

练一练

【练习 8-1】 设计一个通信录结构体类型，包括序号、姓名、性别、年龄、电话和地址。输入一个人的信息，并在屏幕中显示出来。

解：分析该结构体类型中的每个成员，可以将序号（no）定义为整型，姓名（name）定义为字符数组，性别（sex）为字符类型（其中 f 代表女，m 代表男），年龄（age）为整型，电话（phone）为字符数组，地址（address）为字符数组。

源程序如下：

```
struct  message
{  int  no;
   char  name[20];
   char  sex;
   int  age;
   char  phone[20];
   char  address[30];
};
main()
{  struct  message  p;
   printf("\nplease input information of the person:");
   printf("\nno name sex age phone\n");
   scanf("%d %s %c %d %s",&p.no,p.name,&p.sex,&p.age,p.phone);
   getchar();
    /*本行功能为将上行输入的回车符用 getchar();接收*/
   printf("address:");
   gets(p.address);
    printf("The person message is information:");
   printf("\nno:%d",p.no);
   printf("\nname:%s",p.name);
   printf("\nsex:%c",p.sex);
   printf("\nage:%d",p.age);
   printf("\nphone:%s",p.phone);
   printf("\naddress:");
   puts(p.address);
}
```

注意 因为在地址的字符串中可能会有空格，所以不能用 scanf()函数来输入（因为使用 scanf()函数输入字符串时，在遇到 Tab 键、空格或回车符时输入结束），所以用 gets()函数来输入带空格的字符串。

程序运行结果如下：

```
please input information of the person:
no   name  sex  age  phone
1 Wangjin m 21 78342320
address:The east road 12-1

The person message is information:
no:1
name:Wangjin
sex:m
age:21
phone:78342320
address:The east road 12-1
```

本讲小结

本讲主要介绍了结构体类型的基础知识，主要包括：结构体类型的定义方法、结构体类型变量的三种定义方法，结构体类型变量的初始化方法、结构体成员的引用方法、结构体类型的嵌套。

想一想

本讲介绍的学生信息只是一个学生的信息，如果这样的数据要存储多个的话，应该怎样设计存储类型？（提示：我们前面讲过的数组中可以存放多个相同类型的变量。）

第二十讲　结构体数组和指向结构体的指针

学习目标

- 掌握结构体数组的定义和初始化;
- 掌握结构体数组的元素的引用;
- 掌握结构体指针的定义和使用;
- 掌握指向结构体数组的指针的使用方法。

一、结构体数组及指向结构体变量的指针

【思考题 8-2】 如果有多个学生信息（其中每个学生的信息为：学号，姓名，性别，语文、数学、英语成绩），怎样将这些信息存到计算机中去，并进行处理？

程序分析

因为学生是多个，但每个学生的信息格式相同，这样就可以用具有相同数据元素的类型——数组来实现。只不过这次定义数组的类型变成了结构体类型就可以了。多个信息的输入和输出可以用循环语句来实现，下面以这个程序为例，介绍结构体数组的相关知识。这个程序处理了三个学生的信息，并进行显示。

编写程序代码

```
struct  student                 /*定义学生结构体类型*/
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
};
main()
```

```
{  struct  student  stu[3];   /*定义结构体类型的数组，数组中有三个元素*/
   char  ch;
   int  i;
   for(i=0;i<3;i++)
   {  printf("\nplease input information of the student[%d]:",i+1);
      printf("\nnumber,name,sex,chinese,math,english\n");
      scanf("%d",&stu[i].num);
      scanf("%s",stu[i].name);
      getchar();
      /*这里 getchar()是接收输入时在 sex（性别）之前的空格字符*/
      scanf("%c",&stu[i].sex);
      scanf("%d%d%d",&stu[i].chinese,&stu[i].math,&stu[i].english);
   }
printf("\nThe  student information :");
printf("\nnumber    name    sex    chinese   math   english\n");
for(i=0;i<3;i++)
printf("\n%4d%10s%5c%9d%10d%10d",stu[i].num,stu[i].name,stu[i].sex,
stu[i].chinese,stu[i].math,stu[i].english);
}
```



调试运行程序

程序要求从键盘上输入三个学生的学号，姓名，性别（其中 f 代表女，m 代表男）,语文、数学、英语成绩，并将存入的结果显示在屏幕上。



注意　运行程序输入各值的时候，程序中 getchar()函数的作用就是为了接收在输入 name 之后，sex 之前的那个空格；或是每输入一个值后回车时，用来接收回车符。如果没这个函数接收，程序会出错。

程序运行结果如下：

```
please input information of the student[1]:
number,name,sex,chinese,math,english
1 liuli f 98 72 90

please input information of the student[2]:
number,name,sex,chinese,math,english
2 wanghong m 88 79 92

please input information of the student[3]:
number,name,sex,chinese,math,english
3 zhaoqi m 92 94 85

The  student information :
number    name    sex    chinese   math   english

   1     liuli    f       98        72        90
   2  wanghong    m       88        79        92
   3    zhaoqi    m       92        94        85
```



1．结构体数组的定义

与其他基本类型的数组一样，结构体型数组就是定义一组内存空间连续的数据元素，只不过这些数据元素每个都是一个结构体类型的变量。结构体型数组的定义方式如下：

```
struct  结构体类型名
{  成员列表
};
struct  结构体类型名  数组名[数组元素个数];
```

例如在【思考题 8-2】的例程中的定义语句为：

```
struct  student                  /*结构体类型 student 定义*/
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
};
struct  student  stu[3];      /*结构体类型数组的定义语句*/
```

其中 stu 就是 student 结构体类型的结构体数组，在这个数组中有 3 个元素，每个元素都是一个 student 型的结构体变量。数组下标从 0 开始，到 2 结束。

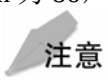


2．结构体数组的初始化

结构体数组也可以像普通数组一样进行初始化，这时可以用两层花括号把初始值括起来，例如：

```
struct  student
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
};
struct student stu[3]={{1,"limei",'f',89,92,87},{2,"wangxing",'m',90,82,95},
                       {3,"Zhaolei",'m',91,86,74}};
```

这时，结构体变量元素 stu[0]中各成员的初始值为：num 为 1，name 为"limei"，sex 为'f'，Chinese 为 89，math 为 92，english 为 87。结构体变量元素 stu[1]中各成员的初始值为：num 为 2，name 为"wangxing"，sex 为'm'，chinese 为 90，math 为 82，english 为 95。结构体变量元素 stu[2]中各成员的初始值为：num 为 3，name 为"Zhaolei"，sex 为'm'，chinese 为 91，math 为 86，english 为 74。



在初始化的时候对应位置的成员赋值与其类型要一致，否则会出错。



3．结构体数组各元素中成员的引用

结构体数组各元素中成员的引用方法如下：

```
数组名[数组下标值].成员名
```

在引用结构体类型数组中各元素时，前面是“数组名[数组下标值]”，然后用直接运算

符“.”连接结构体的“成员名”即可。如【思考题 8-2】的例程中的数组 stu 中下标为 0 的元素的 num 成员，其引用方法为 stu[0].num，为其赋值为 1，写成 stu[0].num=1;即可。



4．指向结构体变量的指针

一个结构体变量的指针就是该变量所占据的内存段的起始地址，可以设一个指针变量，用来指向一个结构体变量，此时该指针变量的值是结构体变量的起始地址，指针变量也可以用来指向结构体数组中的元素。

我们学习指针时讲过，如果想表示一个指针所指向变量的内容，可以用“*”加上指针名来表示，那么一个指向结构体类型的指针就可以表示为：(*指针名).成员名。这里“(*指针名)”等价于“结构体变量名”。但因为“.”的运算优先级别高于“*”，所以必须将“*指针名”用括号括起来才行。

这时就引入了另一个运算符——间接成员运算符（->）。间接成员运算符的左边应是一个结构体指针变量，右边应是该结构体指针变量所指结构体的一个成员，运算结果是一个指针所指结构体中的一个成员变量。间接引用结构体变量成员的一般格式如下：

```
结构体指针变量->成员名
```

它等价于：

```
(*结构体指针变量).成员名
```

例如：

```
struct  student
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
};
struct  student  s1= {1,"limei",'f',89,92,87}, *p;
p=&s1;
```

此时如果用 s1 表示其成员 num，可表示为 s1.num。因为此时 p 指针已指向结构体变量 s1，所以还可用结构体类型指针 p 来间接表示 s1 中各成员，s1 中的 num 成员，可表示为 p->num，它等价于(*p).num，语句：

```
p->num=1;
```

即为 p 指针所指向的结构体变量的 num 成员赋值为 1。

二、指向结构体数组的指针

【思考题 8-3】 如果一个结构体型指针指向了一个结构体型的数组，怎么使用该指针去引用数组中的各元素，如果指针自加或自减时，指针在内存中应该如何移动？



在编写程序中，应该注意一下指针和数组的定义和初始化，以及指针的移动。该例程不使用数组名而直接使用指针来实现各种操作。下面的例程定义了一个结构体类型的数组，里

面存放了三个学生的信息，定义了一个结构体类型的指针，并用指针指向数组首地址，通过指针移动输入各学生的信息。

编写程序代码

```
struct  student
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
};
main()
{  struct student stu[3]={{1,"lilin",'m',89,92,95},{2,"zhangfun",'m',90,87,88},
                          {3,"wangli",'f',99,93,87}},*p;
  int  i;
  printf("\nThe  student information :");
  printf("\nnumber    name    sex    chinese   math   english\n");
  for(p=stu;p<stu+3;p++)
    printf("\n%4d%10s%6c%9d%9d%9d",p->num,p->name,p->sex,p->chinese,p->
           math,p->english);
}
```

调试运行程序

程序运行结果如下：

```
The  student information :
number    name    sex    chinese   math   english

   1     lilin     m       89       92       95
   2  zhangfun     m       90       87       88
   3    wangli     f       99       93       87_
```

小讲堂

5．指向结构体数组的指针

以前已经介绍过，可以使用指向数组及数组元素的指针和指针变量，同样，对结构体数组及其元素也可以用指针或指针变量来指向。

在【思考题 8-3】例程中，就是先让指针指向结构体数组的首地址。在第一次循环中输出 stu[0]的各个成员值。然后执行 p++，使 p 自加 1。p 加 1 意味着 p 所增加的值为结构体数组 stu 的一个元素所占的字节数（在本例中为 2+20+1+2+2+2=29 字节）。执行 p++后 p 的值等于 stu+1，p 指向 stu[1]的起始地址，输出 stu[1]的各成员值，然后 p 再加 1，再输出 stu[2]的各成员值。再执行 p++后，p 的值变为 stu+3，已不再小于 stu+3 了，不再执行循环。

注意以下两点。

（1）如果 p 的初值为 stu，即指向第一个元素，则 p 加 1 后 p 就指向下一个元素的地址。例如：

(++p)->num　　是先使 p 自加 1，然后得到它指向的元素中的 num 成员值。

(p++)->num　　是先得到 p->num 的值，然后使 p 自加 1，指向 stu[1]。

（2）程序定义 p 是指向该结构体类型的指针，即 p 在自加或自减时都是指向结构体数组中每个元素的首地址，则不能将该指针指向结构体的该元素内的某一成员。所以假设 stu 的每个元素占用内存空间为 29 个字节的话，则每次 p 自加或自减时的指针移动字节数都是加减

29 的倍数。例如，下面用法是错误的：

```
p=stu[1].name;
```

即指针 p 只能存放与定义它相同类型的变量地址，而不是什么变量的地址都可以存放。而如下写法是正确的：

```
p=stu;
```

表示 p 指向 stu 数组的首地址。

练一练

【练习 8-2】 用结构体存放表 8-1 中的信息，然后输出每个职工的姓名、基本工资、浮动工资、支出和实发工资（基本工资+浮动工资-支出）。

表 8-1 职工的工资表

姓　名	基 本 工 资	浮 动 工 资	支　出
zhangli	543.0	269.0	78.0
wanghua	655.0	254.0	56.0
zhaoqiao	621.0	312.0	49.0

解：这样的结构可以用结构体类型的数组来实现。因为职工数目为 3 个，所以结构体数组元素个数为 3，结构体类型名为 pay，分析该结构体类型中的每个成员，可以将姓名（name）定义为字符数组，基本工资（wage1）、浮动工资（wage2）、支出（payout）、实发工资（wage3）都定义为单精度浮点型。实发工资在读入一个元素的基本工资、浮动工资和支出后进行计算，结果存入实发工资（wage3）中。数组元素的每个成员可以使用循环语句进行读入。

注意 在读入元素的三个浮点型成员时，必须先用其他简单的单精度浮点型数据读入，然后再赋值给这三个成员，不能直接使用 scanf()函数为这三个成员赋值。

源程序如下：

```
#include "stdio.h"
#include "string.h"
struct pay                    /*定义职工工资结构体类型*/
{ char name[20];
  float wage1,wage2,payout,wage3;
};
main()
{ struct pay wage[3];         /*定义结构体类型的数组，数组中有 3 个元素*/
  int i;
  float w1,w2,w3;
  for(i=0;i<3;i++)
  { printf("please input information of the person wage[%d]:",i+1);
    printf("\nname wage1 wage2 payout\n");
    scanf("%s",wage[i].name);
    scanf("%f%f%f",&w1,&w2,&w3);
    wage[i].wage1=w1;
    wage[i].wage2=w2;
    wage[i].payout=w3;
    /*计算实发工资*/
    wage[i].wage3=wage[i].wage1+wage[i].wage2-wage[i].payout;
```

```
    }
    printf("\nThe  person wage :");
    printf("\n    name      wage1      wage2      payout     realwage ");
    for(i=0;i<3;i++)
    printf("\n%12s%12.1f%12.1f%12.1f%12.1f",wage[i].name,wage[i].wage1,
           wage[i].wage2,wage[i].payout,wage[i].wage3);
}
```

程序运行结果如下：

```
please input information of the person wage[1]:
name wage1 wage2 payout
zhangli 543 269 78
please input information of the person wage[2]:
name wage1 wage2 payout
wanghua 655 254 56
please input information of the person wage[3]:
name wage1 wage2 payout
zhaoqiao 621 312 49

The  person wage :
        name       wage1       wage2       payout      realwage
     zhangli       543.0       269.0        78.0       734.0
     wanghua       655.0       254.0        56.0       853.0
    zhaoqiao       621.0       312.0        49.0       884.0
```

【练习 8-3】 将【练习 8-2】的程序用指向数组的指针来实现。

解：结构体类型及变量的定义同【练习 8-2】，只需要在主函数中定义一个结构体类型的指针 p，初始时将指针 p 指向数组的首地址，然后读入第一个元素的各成员值，再将指针自加，指向下一个元素，再读入第二元素的各成员值，……直到所有元素的成员都输入完毕。

读入完毕后使用公式 p->wage3=p->wage1+p->wage2-p->payout;计算出实发工资，再使用循环语句输出各结构体数组元素的每个成员即可。

源程序如下：

```
#include  "stdio.h"
#include  "string.h"
struct  pay              /*定义职工工资结构体类型*/
{  char  name[20];
   float  wage1,wage2,payout,wage3;
};
main()
{  struct  pay  wage[3],*p;
    /*定义结构体类型数组，数组中有 3 个元素，p 为指向该结构体数组的指针*/
   int  i;
   float  w1,w2,w3;
   for(p=wage;p<wage+3;p++)
  {    printf("please input information of the person wage[%d]:",i+1);
       printf("\nname wage1 wage2 payout\n");
       scanf("%s",p->name);
       scanf("%f%f%f",&w1,&w2,&w3);
       p->wage1=w1;
       p->wage2=w2;
       p->payout=w3;
       p->wage3=p->wage1+p->wage2-p->payout;
   }
  printf("\nThe  person wage :");
  printf("\n\tname \twage1 \twage2 \tpayout \trealwage\n");
  for(p=wage;p<wage+3;p++)
```

```
    printf("\n%8s%8.1f%8.1f%8.1f%8.1f",p->name,p->wage1,p->wage2,p->payout,
           p->wage3);
    }
```

程序运行结果同【练习 8-2】。

本讲小结

本讲主要介绍了结构体数组的定义、结构体数组的初始化、结构体数组元素的引用，指向单个结构体变量的指针和指向结构体数组的指针的使用。在使用结构体类型变量的过程要注意成员的类型与赋值一致性。

想一想

本讲中介绍的结构体数组各元素的成员都是基本数据类型，如果成员是指针，又应该怎样定义和使用？

第二十一讲　结构体与函数

学习目标

- 掌握结构体类型的变量作为函数参数;
- 掌握结构体类型的变量作为函数的返回值。

一、结构体类型的变量作为函数参数

【思考题 8-4】 当使用结构体变量作为函数参数时，我们应该采用什么方式（是值传递还是地址传递）来传递数据？设有一个结构体变量 stu，内含学生学号、姓名和 3 门课成绩，要求在 main 函数中赋值，在另一函数 list 中将它们显示输出。

程序分析

我们在编写该程序时，首先应该确认输入数据的结构体变量定义在哪个函数里，是主函数还是子函数？因为在函数传递过程中，一般是在主函数内输入结构体变量的各成员值，然后对子函数进行调用，所以我们应该将结构体变量定义在主函数内。而结构体的类型定义必须放在所有函数前面定义，这样所有的函数都可以使用该结构体类型来定义变量了。其次注意函数调用语句中实参的类型要与定义的子函数的形参类型一致。

编写程序代码

源程序如下：

```
#include  "string.h"
struct  student
{  int  num;
char  name[20];
int  score[3];
};
void  list(struct  student  stu)    /*定义显示子函数 list*/
{  printf("%d\n%s\n%d\n%d\n%d\n",stu.num,stu.name,stu.score[0],stu.scor
          e[1],stu.score[2]);
   printf("\n");
}
main()
```

```
{  struct  student  stu;
   stu.num=123;
   strcpy(stu.name,"wangli");  /*不能直接将一个字符串通过赋值语句给结构体成员
                                stu.name，必须使用strcpy()字符串拷贝函数赋值*/
   stu.score[0]=89;
   stu.score[1]=92;
   stu.score[2]=75;
   list(stu);                  /*子函数 list 的调用语句*/
}
```



调试运行程序

程序运行结果如下：

```
123
wangli
75
92
3129
```



1．结构体类型的变量作为函数参数

（1）定义结构体类型时，应在子函数和主函数的外面定义，这样才能在两个函数中都使用该类型定义变量。因此，具有多个函数的 C 程序，应将共用的结构体类型定义为全局的，且放在所有函数定义之前，以便所有的函数都可以使用该类型。

（2）子函数的形参为结构体类型，应与主函数中的调用语句中的实参类型相同。

（3）指向结构体的指针变量也可以作为函数的参数。当函数的参数是结构体指针变量时，可以通过结构体指针变量间接地引用其所有成员。

例如【思考题 8-4】的例程可以改为如下的程序：

```
#include  "string.h"
struct  student{int  num;char  name[20];int  score[3];};
void  list(struct  student  *stu)
{  printf("%d\n%s\n%d\n%d\n%d\n",stu->num,stu->name,stu->score[0],stu->
          score[1],stu->score[2]);
   printf("\n");
}
main()
{  struct  student  stu;
   stu.num=123;
   strcpy(stu.name,"wangli");
   stu.score[0]=89;
   stu.score[1]=92;
   stu.score[0]=75;
   list(&stu);
}
```

程序运行结果与原程序一致。

结构体变量的各个成员也可作为函数参数，其使用方法与同类型的普通变量相同。

二、结构体类型的变量作为函数的返回值

【思考题 8-5】 如果想将结构体类型的变量作为函数的返回值，需要注意哪些地方？要求编写两个函数：在函数 input 中输入一个学生信息，在函数 list 中显示其信息。



程序分析

我们可以定义一个学生类型的结构体类型，在这个程序中要求在 input 函数中输入学生信息，然后把这个信息用另一个函数显示出来。在 input 函数里最后要返回输入的学生信息变量到被调用部分，这样就应该在主函数里设一个这样类型的变量来接收 input 函数的返回值。而 list 函数则需要将该变量的值传给 list 的形参并输出结果。



编写程序代码

源程序如下：

```
#define  STUD  struct  student
struct  student
{  int  num;
char  name[20];
int  age ;
char  sex[6];
int  class;
};
STUD  input()                /*定义输入函数 input*/
{  STUD  stu;
   char  ch;
   scanf("%d",&stu.num);
   scanf("%s",stu.name);
   scanf("%d",&stu.age);
   scanf("%s",stu.sex);
   scanf("%d",&stu.class);
   return(stu);
}
void  list(STUD  stu)        /*定义输出显示函数 list*/
{  printf("\nnum:%d",stu.num);
   printf("\nname:%s",stu.name);
   printf("\nage:%d",stu.age);
   printf("\nsex:%s",stu.sex);
   printf("\nclass:%d",stu.class);
}
main()
{  STUD  stu;
   printf("num   name   age  sex  class\n");
   stu=input();
   printf("\n student information\n");
   list(stu);
}
```



调试运行程序

程序运行结果如下：

```
num   name   age  sex  class
1 Limei 18 girl 5

student information:
num:1
name:Limei
age:18
sex:girl
class:5
```



2．结构体类型的变量作为函数返回值

因为前面讲过，函数的类型就是函数的返回值的类型，所以当结构体变量作为函数参数时，在定义该函数时，要保证函数类型名与函数返回值类型一致。

结构体类型的变量作为函数返回值的使用方法与普通变量一致，就是将结构体变量名返回即可。需要注意的是在这种使用方法中，结构体类型的变量的传递方式是值传递方式。

练一练

【练习 8-4】 采用指针传递方式将【思考题 8-2】的例程进行改写，要求函数的形参要用指针表示。

解：改写该程序需要注意以下几点：

（1）结构体类型的指针作为函数参数的格式；

（2）结构体类型的指针作为函数返回值的格式。

源程序如下：

```
#define  STUD  struct  student
struct  student
{  int  num;
   char  name[20];
   int  age ;
   char  sex[6];
   int  class;
};
STUD  *input(STUD  *p)      /*定义输入函数 input*/
{  scanf("%d",&p->num);
   scanf("%s",p->name);
   scanf("%d",&p->age);
   scanf("%s",p->sex);
   scanf("%d",&p->class);
   return(p);
}
void  list(STUD  *p)        /*定义输出显示函数 list*/
{  printf("\nnum:%d",p->num);
   printf("\nname:%s",p->name);
   printf("\nage:%d",p->age);
   printf("\nsex:%s",p->sex);
   printf("\nclass:%d",p->class);
}
main()
{  STUD  stu,*sp;
   printf("num   name   age  sex  class\n");
   sp=input(&stu);
```

```
    printf("\nstudent information:");
    list(sp);
}
```

程序运行结果同【思考题 8-2】的例程结果。

在这个程序中，有两个子函数，一个主函数。其中子函数 input 作用是输入一个结构体变量的各成员信息，子函数 list 的作用是显示结构体变量中的成员信息。

首先，在主函数中定义一个 STUD 类型的结构体变量 stu 和一个结构体类型的指针 sp，然后在主函数中使用函数调用语句 sp=input(&stu);将 stu 的首地址传到子函数 input 中，因为是地址传递，所以在 stu 前加“&”符号。在子函数 input 中，形参为 STUD 类型的指针 p，所以在输入语句中使用“p->成员名”进行输入，存储完成后返回指针 p，就相当于将 p 的地址传回到主函数调用语句中并赋值给 sp。现在 sp 就指向了该结构体变量。

然后，在主函数中使用函数调用语句 list(sp);相当于又将该结构体变量的首地址传给了子函数 list，在子函数 list 中由形参 p 来接收这个地址，再使用 p 来输出各个成员值。

这个程序主要注意的就是结构体类型指针作为函数参数和函数返回值的使用方法。

本讲小结

本讲主要介绍结构体类型的变量作为函数参数，结构体类型的变量作为函数返回值的知识。在结构体变量作为函数参数和函数返回值时，使用方法与普通变量基本一致。只不过这里的结构体变量的复合变量，将其所有成员的值作为一个整体传到子函数的形参中。

想一想

在实用项目中会大量用到结构体类型，函数的类型也可以是结构体变量类型或结构体指针类型，函数的形参和实参也可以是结构体变量类型或结构体指针类型，所以希望能认真学习本章知识，为以后的编程打好基础。请读者思考一下什么时候会用到函数的类型是结构体指针型？怎么定义这样的函数？

第二十二讲　链表

学习目标

- 掌握相关的动态存储分配函数;
- 掌握链表的定义及基本结构;
- 掌握链表的简单操作。

一、链表基础知识及动态分配函数

【思考题 8-6】 前面我们讲过，如果想将学生的信息存入到计算机中，可以设一个结构体类型的数组，但在实际应用过程中，有时很难在开始时就确定下要输入的信息数量。能不能有一种方法，可以不用事先确定存储容量的大小，随心地将每个学生的信息输入进去呢？

C 语言提供给我们一种动态存储分配的数据结构，事先不必确定其长度，且各元素不必顺序存放，各元素之间以指针相互链接。

程序分析

链表是由若干个被称为结点的元素构成的。结点的个数根据需要而定，通常情况下，链

表中的一个结点至少有两个域：一个是数据域（常用 data 表示），用于存储需要处理的数据；另一个是指针域（常用 next 表示），用于存储下一个结点的地址。典型的链表如图 8-2 所示。

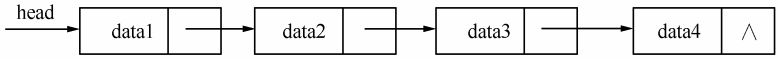


图 8-2　单链表示意图

编写程序代码

链表的结构体类型及指针定义如下：

```
struct  node
{   int  data;                    /*定义数据域*/
    struct  node  *next;          /*定义指针域*/
};
struct  node  *head;              /*定义结构体类型的头指针*/
```

小讲堂

1．链表的基础知识

在图 8-2 中，head 为头指针，该指针指向链表的第一个结点。其中，“∧”表示空指针，在程序中可用常量 NULL 来表示，它表示链表的结束。在链表中，每个结点都没有自己的结构体变量名，是通过 head 头指针找到链表的第一个结点后，用指针指向每个结点来引用其成员的。所以头指针对于链表来说是至关重要的。

小讲堂

2．动态存储分配函数

在定义了 struct node 结构体类型后，并未为其分配空间，链表是在需要时才为结点分配相应的存储单元。C 语言提供了动态分配函数来实现变量的动态存储分配，常用的动态存储分配函数有以下三个。

1）分配内存空间函数 malloc()

（1）malloc()函数的调用格式如下：

```
（类型名*）malloc(分配的内存大小 size)
```

（2）函数功能如下：在内存中分配一个长度为 size 的连续存储空间，返回值是新分配存储空间的首地址，若内存不足，则返回 NULL。其中，“类型名”表示把该存储空间用于何种数据类型，“(类型名*)”表示把 malloc 函数返回值强制转换为该类型指针。例如：

```
int  *p;
p=(int  *)malloc(sizeof(int));
```

程序段表示分配了一个 int 型的内存空间，并强制转换为整型，将 p 指向该空间的首地址。

注意　“sizeof(类型名)”运算符的功能为求出该类型所占内存空间大小，一般在程序中多使用这种方法来计算内存空间大小，而不是直接用数字来指定。

2）分配内存空间函数 calloc()

（1）calloc()函数的调用格式如下：

```
（类型名*）calloc(n,size)
```

（2）函数功能如下：在内存中分配 n 块长度为 size 的连续存储空间，返回值是新分配存储空间的首地址，若内存不足，则返回 NULL。calloc()函数与 malloc()函数均用于动态分配存储空间，区别在于 calloc()函数可以一次分配 n 块区域。例如：

```
char  *p;
p=(char  *)calloc(5,sizeof(char));
```

表示分配 5 个且每个大小为 1 个字节的连续空间，将其类型强制转换为字符类型并赋给 p，结果即让 p 指向该存储空间的首地址。

3）释放内存空间函数 free()

（1）free 函数的调用格式如下：

```
free(指向要释放的空间指针名)
```

（2）函数功能如下：释放该指针所指的一块存储空间，该空间系统可另作它用。注意这个指针所指的空间必须是由 malloc()函数和 calloc()函数分配的才行。free 函数无返回值。例如：

```
int  *pt;                                  /*定义一个整型指针 pt*/
pt=(int  *)malloc(sizeof(int));            /*动态生成一个整型变量并将 pt 指向它*/
free(pt);                                  /*释放 pt 所指的内存单元*/
```

表示释放 pt 指向的存储空间。应注意的是在程序中若使用以上 3 个函数，应在程序的开始处用文件包含命令#include 包含头文件"stdio.h"或"malloc.h"，因版本不同而使用不同的包含文件。

二、链表的操作

【思考题 8-7】 已经了解了链表的结构，那么链表的操作主要有哪些呢？



程序分析

对链表的的操作主要包括链表的建立、链表的输出、链表结点的插入和删除等。下面通过一个完整的例程来学习对链表的各种操作。



编写程序代码

```
#include  <malloc.h>
#define  NULL  0
#define  NODE  struct  node    /*将 struct node 定义成单个标识符 NODE */
struct  node
{  int  data;
   struct  node  *next;
};
/*建立链表的函数 creat*/
NODE  *creat(int  n)
{  NODE  *head,*p,*q;
   int  i;
   if(n==0)
     return(NULL);
   p=(NODE *)malloc(sizeof(NODE));
```

```
   head=p;
   scanf("%d",&p->data);
   p->next=NULL;
   for(i=2;i<=n;i++)
   {  q=(NODE *)malloc(sizeof(NODE));
      scanf("%d",&q->data);
      q->next=NULL;
      p->next=q;
      p=q;
   }
return(head);
}
/*输出链表中各元素函数 out*/
void  out(NODE  *head)
{  NODE  *p;
   p=head;
   if(p!=NULL)
   {  printf("\nThe line element are :");
      while(p!=NULL)
      {  printf("%5d",p->data);
         p=p->next;
      }
   }
}
/*插入链表元素的函数 insert */
NODE  *insert(NODE  *head,int  i,int  x)
{  NODE  *p,*q;
   int  j;
   p=head;
   if(p==NULL)                  /*当链表为空表时*/
   {  q=(NODE *)malloc(sizeof(NODE));
      q->data=x;
      q->next=NULL;
      head=q;
      return(head);
   }
   if(i==1&&p!=NULL)            /*当元素要插入到第一个位置时*/
   {  q=(NODE *)malloc(sizeof(NODE));
      q->data=x;
      q->next=head;
      head=q;
      return(head);
   }
   j=1;
   while(p!=NULL && j<i-1)
   {  p=p->next;
      j++;
   }
   if(p==NULL)
      printf("\nPosition Error!");
   else if(p->next==NULL)        /*当元素插入到表的尾部*/
   {  q=(NODE *)malloc(sizeof(NODE));
```

```
      q->data=x;
      q->next=NULL;
      p->next=q;
      return(head);
    }
    else                              /*当元素插入到中间位置时*/
    {  q=(NODE *)malloc(sizeof(NODE));
      q->data=x;
      q->next=p->next;
      p->next=q;
      return(head);
    }
}
/*删除元素函数 delete*/
NODE  *delete(NODE  *head,int  i,int  *s)
{  int  j;
   NODE  *p,*q;
   p=head;
   if(i<=0)                           /*删除元素的位置错误*/
   {   printf("Delete data position error!");}
   else if(i==1&&p!=NULL)             /*如果删除元素为链表中第一个元素时*/
   {   *s=p->data;
       head=p->next;
       free(p);
       return(head);
   }
   else if(i==1&&p!=NULL&&p->next==NULL)  /*链表中只有一个元素时*/
   {   *s=p->data;
       head=NULL;
       free(p);
       return(NULL);
   }
   j=1;
   while(p!=NULL&&j<i-1)
   { p=p->next;j++; }
     if(p==NULL)
        printf("\n Delete data position error!");
     else if(p->next->next==NULL)   /*删除最后一个元素时*/
        {  q=p->next;
           *s=q->data;
           p->next=NULL;
           free(q);
           return(head);
        }
      else                    /*当插入到中间位置时*/
         {  q=p->next;
            *s=q->data;
             p->next=q->next;
            free(q);
            return(head);
         }
}
```

```
/*在主函数中调用上面各函数实现对链表的操作*/
main()
{  NODE  *head;
   int  n,i,x,flag;
   printf("\nPlease input the number of the creat link element :");
   scanf("%d",&n);
   head=creat(n);
   out(head);
   printf("\nPlease input position and value of the insert element :");
   scanf("%d%d",&i,&x);
   head=insert(head,i,x);
   out(head);
   printf("\nPlease input the position of the delete element :");
   scanf("%d",&i);
   head=delete(head,i,&x);
   if (head!=NULL)
   {  out(head);
      printf("\nThe delete element is :%d",x);
   }
}
```



程序运行结果如下：

```
Please input the number of the creat link element:5
1 2 3 4 5

The line element are:    1    2    3    4    5
Please input position and value of the insert element:4 90

The line element are:    1    2    3   90    4    5
Please input the position of the delete element:2

The line element are:    1    3   90    4    5
The delete element is:2_
```



3．链表的建立

（1）这个函数的形参是要生成的结点个数 n，这个值是由主函数中定义的一个变量传入的。在建立过程中，使用了一个循环，实现多个结点的动态生成。

（2）creat 函数的返回值为指向 struct　node（即 NODE）结构体类型的指针，用于返回新创建的链表的头指针。

（3）在 creat 函数中，定义了三个指向 struct　node 结构体类型的指针：head、p、q。head 为链表的头指针，初始化为 NULL，表示链表为空。q 始终指向新开辟的结点，即要插入的新结点，p 始终指向链表的最后一个结点。新开辟的结点则始终插到链表的最后，即成为 p 所指结点的下一个结点。当循环了 n 次后，结束了链表的建立。

在【思考题 8-7】例程中的建立链表函数 creat 就动态生成一个单链表。



4．链表的输出

（1）out 函数用于链表的输出，无返回值。形参为 struct　node（即 NODE）结构体类型的指针，用来接收链表的头指针。

（2）在 out 函数中，定义了一个 struct　node 结构体类型的指针 p，通过指针 p 输出参数链表中的所有结点的数据域的值。首先，p 指向链表的第一个结点，输出 p 所指结点的数据域的值，然后，p 指向当前结点的后继结点并输出，直到链表结束，此时 p 为 NULL。

在【思考题 8-7】例程中的函数 out 可以输出当前链表的各元素值。



5．链表结点的插入

（1）函数 insert 有三个形参 head、i 和 x。其中 head 为 struct node 结构体类型的头指针，i 表示要插入元素在链表中的位置，x 表示要插入元素的值。

（2）在插入过程中，有以下四种情况：

① 当链表是一个空链表时；

② 当插入到链表的第一个元素之前时（这时需要修改头指针）；

③ 当链表非空，插入位置为链表的尾部时；

④ 当链表非空，插入到链表中间（前后都有元素）时。

在【思考题 8-7】例程中函数 insert 可以在链表的某个位置插入一个元素。



6．链表结点的删除

（1）函数 delete 有两个形参 head 和 i，其中 head 为 struct node 结构体类型的头指针，i 表示要删除元素在链表中的位置。

（2）delete 函数的返回值为 struct node 结构体类型指针，用于将操作后的链表新的头指针返回。

（3）在删除元素过程中，有以下四种情况：

① 当链表是一个空链表时，删除出错；

② 当删除的是链表的第一个元素时（这时需要修改头指针）；

③ 当链表非空，删除的位置为链表的最后一个元素时；

④ 当链表非空，删除的是链表中间元素时。

在【思考题 8-7】例程中函数 delete 可以删除链表某个位置的一个元素。



7．主函数的实现

（1）在主函数中，实现了对以上四个子函数的调用，在调用过程中，需要注意各个函数的形参类型和返回值类型。

（2）在主函数中，我们是采用了建立——输出结果，插入元素——输出结果，删除元素——输出结果的过程，将每个函数调用一遍。

在【思考题 8-7】例程中的主函数 main 实现了将上述各子函数调用的功能，完成了链表的各种操作。

练一练

【练习 8-5】 设计一个程序，输入 n 个学生成绩（学号和成绩），然后输出需要补考的学生的信息。要求用单链表实现。

解：程序分析如下：

（1）定义一个结构体类型 struct　student 包含 3 个成员：num、score 和指针域 next。链表结点为 struct student 类型。

（2）函数 creat_list()的功能是创建学生成绩链表，方法同前面讲述的链表建立函数。形参 n 表示创建包括 n 个结点的类型。

（3）函数 delete_list()的功能是求需要补考的学生信息。函数体中将成绩 score>=60 的学生成绩删除，剩余的链表中结点即为需要补考的学生信息。

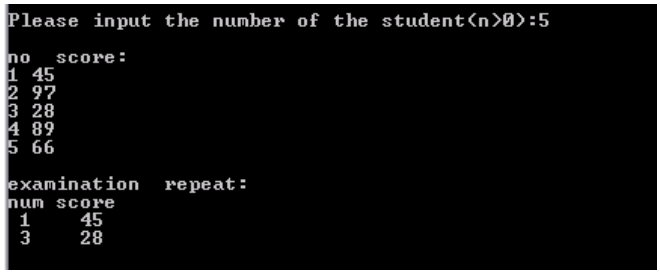
（4）函数 print()用来输出补考学生的信息。

源程序如下：

```
#include "stdlib.h"
#define STU struct student
struct student                          /*定义学生结点类型*/
{  int  num;
   int  score;
   struct  student  *next;
};
STU  *creat_list(int  n)
{  STU  *head,*new,*p;                  /*new 指向每次新开辟的结点，p 为工作指针*/
   int  i;
   new=(STU  *)malloc(sizeof(STU));     /*开辟新空间并由 new 指向该结点*/
   printf("\nno  score:\n");
   scanf("%d%d",&new->num,&new->score);
   head=p=new;
   for(i=1;i<n;i++)
   {  new=(STU *)malloc(sizeof(STU));
      scanf("%d%d",&new->num,&new->score);
      p->next=new;
      p=new;
   }
   p->next=NULL;
   return(head);
}
STU  *delete_list(STU  *head)
{  STU  *p=head,*q=head;
   if(head==NULL)  return(head);
   while(p!=NULL)
   {
      if(head->score>=60)
      {  head=head->next;
```

```
            free(p);
        }
        else
        {  if(p->score>=60)
             {  q->next=p->next;
                free(p);
               p=q->next;
             }
             else
            {  q=p;
               p=p->next;
            }
        }
     }
    return(head);
}
void  print(STU  *head)
{  STU  *p;
   p=head;
   if(head==NULL)
      printf("\nscore<60 is not found!");
   else
   {  printf("\nexamination  repeat:\n");
      printf("num score\n");
      while(p!=NULL)
      {  printf("%2d%6d\n",p->num,p->score);
         p=p->next;
      }
   }
}
main()
{  STU  *head;
   int  n;
   printf("\nPlease input the number of the student(n>0):");
   scanf("%d",&n);
   head=creat_list(n);
   head=delete_list(head);
   print(head);
}
```

程序运行结果如下：



本讲小结

本讲是结构体类型应用的一个提高。在实际使用中，链表的应用也十分广泛，在这里，我们只做简单介绍，在数据结构课程中大家会继续学习更深入的知识。

想一想

我们这里只介绍了一种链表形式，就是每个结点只有一个指针域的单链表形式，实际上为了方便操作，我们还可以设计双向链表和循环链表等多种形式的链表，具体形式主要由实际需要来确定。请读者自己考虑一下双向链表和循环链表的定义格式和操作方法是怎样实现的？

第二十三讲　共用体、枚举、typedef类型定义



学习目标

- 掌握共用体的定义与使用；
- 掌握共用体与结构体类型的区别；
- 掌握枚举类型的定义与使用；
- 掌握 typedef 类型的定义及应用。

一、共用体类型的定义与使用

【思考题 8-8】 如果程序运行的内存空间很小，要求我们将不同的数据类型放在同一内存空间中，不同的时候使用不同的类型。这样的使用方法能否实现？

C 语言就提供了这样一种数据类型，它与结构体类型相似，但在使用过程中却有不同之处。请看下面的例程。



程序分析

我们设计一个教师与学生通用的结构体类型，教师信息有姓名、年龄、职业、教研室四项。学生信息有姓名、年龄、职业、班级四项。设计一个程序，输入老师或学生信息，然后显示出来。



编写程序代码

源程序如下：

```
#include "stdio.h"
union depart
{ int class;
  char office[20];
};
struct person
{ char name[10];
  int age;
  char job;
  union depart depa;
};
main()
{ struct person person[2];
```

```
    int  n,i;
    for(i=0;i<2;i++)
    {  printf("input name,age,job and department:\n");
       scanf("%s %d %c",person[i].name,&person[i].age,&person[i].job);
       if(person[i].job=='s')
          scanf("%d",&person[i].depa.class);
       else
          scanf("%s",person[i].depa.office);
     }
     printf("name\tage\t job    class/office\n");
     for(i=0;i<2;i++)
     {  if(person[i].job=='s')
        printf("%s\t %3d %5c\t%d\n",person[i].name,person[i].age,
               person[i].job,person[i].depa.class);
        else
           printf("%s\t %3d %5c\t%s\n",person[i].name,person[i].age,
                  person[i].job,person[i].depa.office);
     }
}
```



调试运行程序

程序运行结果如下：

```
input name,age,job and department:
liuli 19 s 13
input name,age,job and department:
wanghua 32 t computer

name    age      job    class/office
liuli     19      s      13
wanghua   32      t      computer
```



共用体也是一种构造数据类型，它将不同类型的数据项存放在同一个内存区域内，组成共用体的各个数据项也称为成员或域，共用体也称为联合。

共用体与结构体的不同之处在于，结构体变量的各成员占用的是连续的不同的存储单元，而共用体变量的各成员占用的是相同的存储单元。由于共用体类型将不同类型的数据在不同时刻存储到同一内存区域内，因此使用共用体数据可以更好地利用存储空间。

1．共用体类型的定义

共用体类型定义一般格式如下：

```
union   共用体类型名
{   成员列表
};
```

其中，union 是关键字，表示后面的类型是一个共用体类型，不能省略。共用体类型名为合法的标识符，花括号内的成员列表用来说明组成该共用体的各个成员，对每个成员应进行类型说明。

其成员定义方法与其他简单类型变量一致，其说明格式为：

```
类型符  成员名;
```

例如：

```
union data
{  int  a;
   float  b;
   char  c;
};
```

上面语句定义了一个共用体类型 data，里面有三个成员 a、b、c。a 为一个整型变量，b 为一个单精度浮点型变量，c 为一个字符型变量，它们共用同一段内存空间。共用体类型的长度等于最长的成员的长度，而结构体类型的长度是其所有成员长度之和。



2．共用体变量的定义

共用体变量的定义和结构体变量的定义方式相似，也有三种方法。

1）先定义共用体类型后定义共用体变量

```
union  共用体类型名
{  成员列表
};
union  共用体类型名  变量名表;
```

例如：

```
union  data
{  int  a;
   float  b;
   char  c;
};
union  data  x,y;
```

这种定义方法就是先定义出共用体类型，再使用 union 和共用体名再加上多个变量的定义方法。这种方法是通常的定义方法。也可以用#define 定义一个符号常量来代表一个共用体类型，例如：

```
#define  DATA  union  data;
```

这时，就可以用 DATA 来代替 union data 来定义变量：

```
DATA  x,y;
```

2）在定义共用体类型的同时定义结构变量

```
union  共用体名
{  成员列表
}变量名表;
```

例如上面的定义可改为：

```
union  data
{  int  a;
   float  b;
```

```
    char  c;
}x,y;
```

这种定义方法可以在定义类型的同时直接定义出各个变量。

3）直接定义共用体变量

```
union
{   成员列表
}变量名表;
```

例如上面的定义还可改为：

```
union
{  int  a;
   float  b;
   char  c;
}x,y;
```

这种方法可以不指明共用体类型名而直接定义出各个变量，但有一个缺点，就是这种定义方法只能在定义类型的同时直接定义出变量，在以后程序其他位置就因为没有共用体类型名而无法再定义其他变量。



3．共用体成员的引用

定义了共用体变量后，就可使用它。但共用体变量与结构体变量不同的是，不能在定义的同时初始化，但可对第一个成员赋初值。

例如：下面的两个定义中，第一个是合法的，第二个是不合法的。

```
union  data  x={10};
union  data  x={10,23.5, 'a'};
```

对共用体变量的使用是通过对其成员的引用实现的，引用共用体变量成员一般格式如下：

```
共用体变量名.成员名
```

例如，给共用体变量 x 的 a 成员赋值为 2，语句为：

```
x.a=2;
```

应注意的是，对于一个共用体变量来说，每次只能给一个成员赋值，不能同时给多个成员赋值。

共用体成员的特点如下。

（1）共用体变量的地址和其名成员的地址相同。

（2）共用体变量不能整体赋值，不能整体输入和输出，不能作为函数参数，也不能作为函数的返回值，但可以使用指向共用体变量的指针。

（3）可在共用体变量的内存空间中存放不同类型的成员，但在任一时刻只能存放其中的一个成员，而不能同时存放所有的成员。

（4）共用体变量中起作用的是最近一次存放的成员，在存入新成员后，原有的成员就失去了作用。

（5）共用体变量可以出现在结构体类型中，结构体类型的变量也可以出现在共用体类型

中，即两者可以相互嵌套。

二、枚举类型的定义与使用

【思考题 8-9】 若某个变量只能取少数几个值，可否将其允许取值一一列出，定义后该类型的变量只能取列出的若干个值之一？

C 语言提供给我们“枚举”类型，就是将变量的取值指定为若干值之一，其中每个值用一个名字标识。



程序分析

下面程序的功能是从键盘中输入一个 1～7 之间的整数，并把它转换为星期一到星期日的各个英文单词显示。注意枚举类型的定义及使用方法。



编写程序代码

```
#include  "stdio.h"
main()
{  enum  week{MON=1,TUE,WED,THU,FRI,SAT,SUN};
   enum  week  day;
   int  i;
   printf("input i:");
   scanf("%d",&i);
   day=(enum week)i;
   switch (day)
   { case MON: printf("\nMONDAY\n");     break;
     case TUE: printf("\nTUESDAY\n");    break;
     case WED: printf("\nWEDNESDAY\n");  break;
     case THU: printf("\nTHURSDAY\n");   break;
     case FRI: printf("\nFRIDAY\n");     break;
     case SAT: printf("\nSATURDAY\n");   break;
     case SUN: printf("\nSUNDAY\n");     break;
     default:  printf("\ninput error!\ninput inter(0~7)!\n");
   }
}
```



调试运行程序

程序运行结果如下：

```
input i:6
SATURDAY
```



4．枚举类型的定义

枚举类型定义的一般格式如下：

```
enum  枚举类型名  {枚举值表};
```

或写为：

```
enum  枚举类型名
{     枚举值表
};
```

其中，enum 为关键字，表示定义一个枚举类型。枚举类型名须为 C 语言合法的标识符。花括号内的标识符称为枚举元素或枚举常量，各枚举常量之间用逗号隔开，注意右花括号后的分号“;”不能省略。例如，定义一个枚举类型 week 代表一周的七天，定义格式如下：

```
enum  week {MON,TUE,WED,THU,FRI,SAT,SUN};
```

应注意的是，每个枚举常量对应着一个整数值。一般情况一下，枚举类型中的枚举常量是从 0 开始顺序取值的。如上面语句中，从 MON 到 SUN 的取值分别为 0、1、2、3、4、5、6。

我们也可以显式指定各枚举常量的值，如：

```
enum  week {MON=1,TUE,WED,THU,FRI,SAT,SUN};
```

这时从 MON 到 SUN 的取值分别为 1、2、3、4、5、6、7，每个值顺序加 1。我们也可以分隔着定义各常量的值，这时在显式定义的值后面的各隐式的值是显式的值依次加 1，直到下一个显式的值改变，如：

```
enum  color {red=0,yellow,blue=3,white,black};
```

此时，red=0，则 red 之后的 yellow 顺序增 1，yellow 为 1；同理 blue=3，则 white 为 4，black 为 5。



5．枚举型变量的定义

（1）先定义枚举类型后定义枚举型变量。与结构体或共用体类型变量定义的基本方法相似，这种方法就是先定义枚举类型，然后使用“enum　枚举类型名”来定义这种类型的变量，例如：

```
enum  week {MON,TUE,WED,THU,FRI,SAT,SUN};
enum  week  day;     /*定义 week 类型的枚举变量 day*/
```

（2）在定义枚举类型的同时定义枚举型变量。这种方法是在定义枚举类型的后面直接定义出该类型的变量，可以简化程序。

```
enum  week { MON, TUE, WED, THU, FRI, SAT, SUN }day;
```

（3）直接定义枚举类型变量。这种定义方法可以省略枚举类型名，直接定义出枚举变量。但这种方法不能在其他位置再定义这种枚举类型的变量。

```
enum {  MON,TUE,WED,THU,FRI,SAT,SUN }day;
```



6．枚举变量的使用

枚举型变量只能取相应枚举类型列表中的各值，如：

```
enum  week  {MON=1,TUE,WED,THU,FRI,SAT,SUN};
enum  week  day;
day=WED;
```

变量 day 为 week 型的枚举变量。



（1）在枚举类型定义中，枚举常量的命名规则与标识符相同，并且不能另作它用。

（2）枚举常量不是变量，不能在程序中用赋值语句对其赋值。

（3）只能把枚举常量赋给枚举变量，不能把对应的整数直接赋给枚举变量，但可以用强制类型转换来进行转换。如，day=7;语句是错误的，但 day=(enum　week)7;语句则是正确的。

（4）枚举常量不是字符常量也不是字符串常量，使用时不能加单、双引号。

（5）输出枚举常量或枚举变量的整数值时，应使用整型输出格式符。若要输出枚举常量名，需经过转换。可以用以下语句输出其常量名。

```
day=MON;
if(day==MON)
printf("MONDAY");
```

（6）枚举常量可以进行比较运算，由它们对应的整数参加比较。

三、typedef类型的定义及应用

【思考题 8-10】 在我们学习链表的知识时，定义结构体类型的指针时，每次都需使用 struct 加上结构体名来定义，十分麻烦。C 语言可以将这种书写复杂的类型名重新定义，变成一个像整型、浮点型那样的简单定义格式吗？

C 语言提供给用户一个重新定义类型的语句，那就是 typedef 语句，使用它，可以将复杂的构造类型重新定义成一个简单类型，这样书写的时候十分方便。



使用 typedef 语句将结构体类型 struct　student 重新定义为 STUD 类型。




```
typedef  struct student
{  int  num;
   char  name[20];
   char  sex;
   int  chinese,math,english;
}STUD;
```



7．类型定义的基本格式

typedef　原类型名　新类型名;

其中，typedef 为关键字，表示重定义。原类型名是 C 语言提供的任一种数据类型，可以是简单数据类型，也可以是构造数据类型；新类型名是代表原类型名的一个别名。使用新类型名可以像使用原类型名那样定义变量了。例如：

```
typedef  int  integer;
integer  x,y;
```

其中第二条语句等价于：

```
int  x,y;
```

再如，原来有一个结构体类型 birthday，及结构类型及变量定义如下：

```
struct  birthday
{  int  year;
   int  month;
   int  day;
};
struct  birthday  day;
```

现在使用 typedef 语句就可以重定义为如下格式：

```
typedef  struct  birthday
{  int  year;
   int  month;
   int  day;
}BIR;
BIR  day;
```

其中 struct　birthday　day;和 BIR　day;这两个语句是等价的。这样就简化了程序的书写，所以在大量使用结构体类型的程序中大多采用 typedef 语句对结构体类型进行重定义。



8．类型定义的使用说明

（1）使用 typedef 只是为已经存在的类型定义一个别名，并没有产生一个新的类型，原来的型名仍可用。

（2）使用 typedef 类型定义可以定义各种类型别名，但不能定义变量。

（3）typedef 与#define 比较，如：

```
typedef  char  CH;
#define  char  CH;
```

两者的作用都是用 CH 来代表 char，但 typedef 是在编译时处理的，不是进行简单字符替换，而是采用新类型名定义变量。而#define 是在预编译时处理，只进行简单字符串替换。

练一练

【练习 8-6】 从键盘输入一个十六进制数，交换该十六进制数的高位和低位，并将交换后的结果显示。

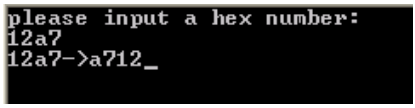
解：由题意可知要交换十六进制数的高位和低位，最好采用的是共用体类型，一个成员是整型，占用 2 个字节。因为字符型变量在内存中占用字节数为 1 个字节，且存放的是字符的 ASCII 码，所以另一个成员设为结构体类型成员，结构体类型中有 2 个字符型的变量即可。

源程序如下：

```
#include  "stdio.h"
main()
{  union  u
   {  struct  byte
      {  unsigned  char  l;
         unsigned  char  h;
      }byte;
      unsigned  int  data;
    }u1,u2;
    printf("please  input  a  hex  number:\n");
    scanf("%x",&u1.data);
    u2.byte.l=u1.byte.h;
    u2.byte.h=u1.byte.l;                    /*交换高低位*/
```

```
        printf("%x->%x",u1.data,u2.data);
    }
```

程序运行结果为：



本讲小结

本讲主要介绍了共用体类型及其变量的定义和使用方法，枚举类型及其变量的定义和引用，typedef 类型定义的方法。需要掌握共用体和结构体类型的区别、枚举类型的特点、typedef 类型定义的格式。

想一想

在编写程序时什么时候用结构体类型，什么时候用共用体类型？什么时候会用到枚举类型？使用 typedef 类型重定义有什么好处？

本章小结

本章主要介绍了结构体、共用体和枚举类型，这些都是复合的数据类型，通过这些类型的学习，使读者掌握较复杂的数据结构。还介绍了用 typedef 类型定义的语句，它可以使程序简化，方便编程。

结构体类型将不同类型的数据组合在一起，方便数据的处理。使用结构体类型时，必须先定义类型，再定义结构体变量后才能使用。

通过直接成员运算符“.”或间接引用成员运算符“->”来引用结构体变量的各个成员，其成员的使用与同类型普通变量相同。结构体变量可以作为一个整体参加赋值运算，但不能作为一个整体进行输入/输出。

可以将结构体变量作为函数参数传递，也可以作为函数的返回值，其使用与其他数据类型相似，实参与形参类型应保持一致。

共用体也是一种构造类型，它的各个成员占有同一段存储空间。共用体变量所占的空间大小等于其成员中所占存储空间的最大数。使用共用体类型的目的是节省内存。

枚举类型是一种用户自定义的数据类型，在枚举类型的定义中列举出所有可能的取值，被声明为该枚举类型的变量其取值不能超过定义的范围。

typedef 类型定义只是定义了一个数据类型的别名，而不是定义一种新的数据类型，所以原来的类型名仍可以使用，不能使用 typedef 定义变量。

课后习题八

一、选择题

1．当说明一个结构体变量时系统分配给它的内存是_____。

A）各成员所需内存量的总和　　B）结构中第一个成员所需内存量

C）成员中占内存量最大者所需的容量　　D）结构体中最后一个成员所需内存量

2．设有以下语句：

```
struct stu
{ int a;
  float b;
}stutype;
```

则下面的叙述不正确的是_____。

A）struct 是结构体类型的关键字　　B）struct stu 是用户定义的结构体类型

C）stutype 是用户定义的结构体类型名　　D）a 和 b 都是结构体成员名

3．若有如下定义：

```
struct data
{ int i;
  char ch;
  double f;
}b;
```

则结构体变量 b 占用内存的字节数是_____。

A）1　　B）2　　C）8　　D）11

4．以下程序的运行结果是_____。

```
#include "stdio.h"
main()
{ struct date
  { int year,month,day;
  }today;
  printf("%d\n",sizeof(struct date));
}
```

A）6　　B）8　　C）10　　D）12

5．设有如下定义：

```
struct sk
{ int n;
  float x;
}data,*p;
```

若要使 p 指向 data 中的 n 域，正确的赋值语句是_____。

A）p=&data.n;　　B）*p=data.n;

C）p=(struct sk *)&data.n;　　D）p=(struct sk *)data.n;

6．若有以下说明和语句：

```
struct student
{ int age;
  int num;
}std,*p;
p=&std;
```

则以下对结构体变量 std 中成员 age 的引用方式不正确的是_____。

A）std.age　　B）p->age　　C）(*p).age　　D）*p.age

7．若有以下说明和语句，则 scanf 函数调用语句中对结构体变量成员引用方式不正确的是_____。

```
struct pupil
{ char name[20];
  int age;
  int sex;
}pup[5],*p;
p=pup;
```

A）scanf("%s",pup[0].name);　　B）scanf("%d",&pup[0].age);

C）scanf("%d",p->sex);　　D）scanf("%d",&(p->age));

8．以下程序的输出结果是_____。

```
struct stu
{ int x;
  int *y;
}*p;
int dt[4]={10,20,30,40};
struct stu a[4]={50,&dt[0],60,&dt[1],70,&dt[2],80,&dt[3]};
main()
{ p=a;
  printf("%d",++p->x);
  printf("%d",(++p)->x);
  printf("%d\n",++(*p->y));
}
```

A）10,20,20　　B）50,60,21　　C）51,60,21　　D）60,70,31

9．有如下定义：

```
struct person
{ char name[9];
  int age;
};
struct person class[10]={"Johu", 17, "Paul", 19, "Mary", 18, "Adam", 16};
```

根据上述定义，能输出字母 M 的语句是_____。

A）prinft("%c\n",class[3].mane);　　B）pfintf("%c\n",class[3].name[1]);

C）prinft("%c\n",class[2].name[1]);　　D）printf("%c\n",class[2].name[0]);

10．下列程序的输出结果是_____。

```
struct abc
{ int a,b,c; };
main()
{ struct abc s[2]={{1,2,3},{4,5,6}};
  int t;
  t=s[0].a+s[1].b;
  printf("%d \n",t);
}
```

A）5　　B）6　　C）7　　D）8

11．以下对 C 语言中共用体类型数据的叙述正确的是_____。

A）可以对共用体变量直接赋值

B）一个共用体变量中可以同时存放其所有成员

C）一个共用体变量中不能同时存放其所有成员

D）共用体类型定义中不能出现结构体类型的成员

12．设有如下枚举类型定义：

```
enum language { Basic=3,Assembly,Ada=100,COBOL,Fortran};
```

则枚举量 Fortran 的值为_____。

A）4　　B）7　　C）102　　D）103

13．以下各选项为说明一种用户自定义类型名，其中正确的是_____。

A）typedef v1 int;　　B）typedef v2=int;

C）typedef int v3;　　D）typedef v4:int;

14．下面对 typedef 的叙述中不正确的是_____。

A）用 typedef 可以定义各种类型名，但不能用来定义变量

B）用 typedef 可以增加新类型

C）用 typedef 只是将已存在的类型用一个新标识符来代表

D）使用 typedef 有利于程序的通用和移植

15．以下程序的运行结果是_____。

```
typedef union
{ long a[2];
  int b[4];
  char c[8];
}TY;
TY our;
main()
{ printf("%d\n",sizeof(our)); }
```

A）32　　B）16　　C）8　　D）24

二、填空题

1．以下程序的运行结果是_____。

```
struct n
{ int x;
  char c;
};
main()
{ struct n a={10,'x'};
  func(a);
  printf("%d,%c",a.x,a.c);
}
func(struct n b)
{ b.x=20;
  b.c='y';
}
```

2．以下程序的运行结果是_____。

```
main()
{ struct EXAMPLE{ struct
                  { int x;
                    int y;
                  }in;
                   int a;
                   int b;
                 }e;
  e.a=1;e.b=2;
  e.in.x=e.a*e.b;
  e.in.y=e.a+e.b;
  printf("%d,%d",e.in.x,e.in.y);
}
```

3．以下程序用以输出结构体变量 bt 所占内存单元的字节数，请在_____内填上适当内容。

```
struct  ps
{  double  i;
   char  arr[20];
};
main()
{  struct  ps  bt;
   printf("bt  size:%d\n",_____________________);
}
```

4．设有三个人的姓名和年龄存在结构数组中，以下程序输出三人中年龄居中者的姓名和年龄，请在_____内填上正确内容。

```
static  struct  man
{  char  name[20];
   int  age;
}person[]={ "li-ming",18,"wang-hua",19,"zhang-ping",20 };
main()
{  int  i,j,max,min;
   max=min=person[0].age;
   for(i=1;i<3;i++)
     if(person[i].age>max)    【1】 ;
     else  if(person[i].age<min)   【2】 ;
   for(i=0;i<3;i++)
     if(person[i].age!=max 【3】 person[i].age!=min)
      {  printf("%s %d\n",person[i].name,person[i].age);
         break;
      }
}
```

【1】_____【2】_____【3】_____

5．写出下列程序的输出结果是_____。

```
enum  coin{penny,nickel,dime,quarter,half_dollar,dollar};
char *name[]={"penny","nickel","dime","quarter","hal_fdollar","dollar"};
main()
{   enum  coin  money1,money2;
    money1=dime;
    money2=dollar;
    printf("%d,%d\n",money1,money2);
    printf("%s,%s\n"),name[(int)money1],name[(int)money2]);
}
```

三、编程题

1．试利用结构体类型编制一程序，实现输入一个学生数学的期中成绩和期末成绩，然后计算并输出其平均成绩。

2．有一个 unsigned long 型整数，现要分别将其前 2 个字节和后 2 个字节作为 2 个 unsigned int 型整数输出（设一个 int 型数据占 2 个字节），试编一函数 partition 实现上述要求。要求在主函数中输入该 long 型整数，在函数 partition 中输出结果。

3．请定义枚举类型 money，用枚举元素代表人民币的面值，包括 1、2、5 分；1、2、5 角；1、2、5、10、50、100 元。

第九章　位　运　算

计算机真正执行的是由 0 和 1 信号组成的计算机指令，数据也是以二进制表示的。因此最终要实现计算机的操作，就要对这些 0 和 1 信号进行操作。每一个 0 和 1 的状态称为一个“位”（bit）的状态。有了位运算，C 语言就能编写出直接对计算机硬件进行操作的程序，具有汇编语言的一些功能。

第二十四讲　位运算

学习目标

- 掌握各种位运算符的功能、运算规则及使用方法，能够应用各种位运算符解决实际问题;
- 理解并掌握各种复合赋值位运算符的功能及应用;
- 了解位段的有关知识。

一、位运算

【思考题 9-1】 设有 a、b 两数，分别求出“a&b”、“a|b”、“a∧b”、“～a”、“a<<2”及“a>>2”的结果。

程序分析

位运算的运算符一共有 6 个，是对二进制数位进行的运算。在这里我们应该对什么是位运算，以及位运算的操作符都代表什么意义进行研究。

编写程序代码

```
#include "stdio.h"
main()
{ unsigned char a,b;                    /*定义两个无符号字符型变量 a，b*/
  unsigned char c1,c2,c3,c4,c5,c6;/*定义 c1~c6 六个变量，存放结果*/
  printf("enter a and b:");
  scanf("%d%d",&a,&b);                  /*输入十进制数 a 与 b 的值*/
  c1=a & b;                             /*对 a 与 b 进行与运算，结果赋值给 c1*/
  c2=a | b;                             /*对 a 与 b 进行或运算，结果赋值给 c2*/
  c3=a∧b;                               /*对 a 与 b 进行异或运算，结果赋值给 c3*/
  c4=～a;                               /*对变量 a 全部位取反，结果赋值给 c4*/
  c5=a<<2;                              /*a 的各位全部左移 2 位，结果赋值给 c5*/
  c6=a>>2;                              /*a 的各位全部右移 2 位，结果赋值给 c6*/
  printf("a & b =%d\n",c1);             /*输出十进制数 c1 值*/
  printf("a | b =%d\n",c2);             /*输出十进制数 c2 值*/
  printf("a ∧b =%d\n",c3);              /*输出十进制数 c3 值*/
  printf("～a    =%d\n",c4);            /*输出十进制数 c4 值*/
  printf("a<<2  =%d\n",c5);             /*输出十进制数 c5 值*/
```

```
    printf("a>>2  =%d\n",c6);          /*输出十进制数 c6 值*/
}
```



调试运行程序

程序运行结果如下：

```
enter a and b:12 23
a & b =4
a | b =31
a ^ b =27
~a    =243
a<<2  =48
a>>2  =3
```



1．位运算符

所谓位运算，是指对二进制数位进行的运算。每一个二进制位只能存放 0 或 1，因此位运算符的运算对象是一个二进制数位的集合。例如，将一个存储单元中的各二进制位左移或右移一位，两个数按位相加等。C 语言提供的位运算符如表 9-1 所示。

表 9-1　C 语言的算术运算符

运　算　符	名　　称	运 算 类 型	示　　例	功　　能
～	按位取反	单目运算符	～a	对变量 a 的全部位取反
<<	左移	双目运算符	a<<2	a 的各位全部左移 2 位
>>	右移	双目运算符	a>>2	a 的各位全部右移 2 位
&	按位与	双目运算符	a&b	a 和 b 的各位按位进行“与”运算
\|	按位或	双目运算符	a\|b	a 和 b 的各位按位进行“或”运算
∧	按位异或	双目运算符	a∧b	a 和 b 的各位按位进行“异或”运算

说明：

（1）参与位运算的运算对象必须是整型或字符型数据，不能是其他类型的数据。

（2）两个长度不同的数据进行位运算时，系统先将两者的右端对齐，短的运算对象若是有符号数则按符号位扩展，若是无符号数则以“0”扩充。

（3）除了按位取反运算符“～”的结合方向是自右至左外，其余位运算符的结合方向都是自左至右。

（4）位运算符的优先级如下：按位取反运算符“～”的优先级最高，高于所有的双目运算符；其次是左移运算符“<<”和右移运算符“>>”，其优先级高于关系运算符；最低的是按位与“&”、按位异或“∧”和按位或“|”运算符，其优先级低于关系运算符。

位运算符与赋值运算符结合可以形成复合赋值位运算符，可以参见第三讲中的表 2-3。



2．位运算

1）按位取反运算

按位取反运算符用“～”表示，是位运算中唯一的单目运算符，运算对象应置于运算符的右边。按位取反运算的运算规则是：把运算对象的内容按位取反，将每一位上的 0 变 1，1

变 0，即～0=1，～1=0。

例如：a=0000 0000 0000 1011，则表达式～a 的值为 1111 1111 1111 0100。

在【思考题 9-1】中，语句：

```
…
unsigned  char  a,b;          /*定义两个无符号字符型变量 a，b*/
…
c4=～a;                       /*对变量 a 全部位取反，结果赋值给 c4*/
…
printf("～a =%d\n",c4);       /*输出十进制数 c4 值*/
…
```

代码：c4=～a;，就是一个按位取反运算，当输入 a 的值为 12 时，则输出显示结果是 243。这是因为，a 为 unsigned char 数据类型，内存单元个数是 8 位，而输入的 a 为十进制数 12，12 的八位二进制编码是 0000 1100，将其进行按位取反，得二进制 1111 0011，将其转换为十进制数为 243。

按位取反的运算的应用：常用于将一个数据的全部数位翻转。

2）按位与运算

按位与运算用“&”表示，规则是：若两个运算对象的对应二进制数位均为1，则结果的对应数位为 1，否则为 0。按位与运算可能的运算组合及其运算结果如下所示：

0&0=0　　1&0=0　　0&1=0　　1&1=1

按位与运算的特点：二进制数的任何数位，只要和数位 0 进行“与”运算，该位清零；和数位 1 进行“与”运算，该位保留原值不变。

例如，x=0000 0000 0000 1011，y=0000 0000 0000 1010，则表达式 x&y 的计算结果如下：

```
  0000 0000 0000 1011
(&) 0000 0000 0000 1010
  ---------------------
  0000 0000 0000 1010
```

在【思考题 9-1】中，语句：

```
c1=a & b;      /*对 a 与 b 进行与运算，结果赋值给 c1*/
```

就是一个按位与的运算，当输入整数 a、b 的值分别为 12 和 23 时，系统将其自动转换为二进制数，代码 a & b 相当于 0000 1100 & 0001 0111，最后得到二进制编码为 0000 0100，转换成十进制数为 4。

按位与运算的应用如下。

（1）将数据中的某些位清零。如原有数为 0000 0000 0000 1011，要将其第 1 位和第 2 位清零，只需将该数要清零的位数设为 0，其余数位写 1，即与 1111 1111 1111 1000 进行与运算即可。

（2）取一个数据的某些数位。如原数为 0000 0000 0000 1011，若只保留第 2 位和第 3 位，只需将相应的位数设为 1，其余数位写 0，即与 0000 0000 0000 0110 进行与运算即可。

3）按位或运算

按位或运算符用“|”表示，规则是：若两个运算对象的对应二进制位均为 0，则结果的对应数位为 0，否则为 1。按位或运算可能的运算组合及其运算结果如下所示：

0|0=0　　1|0=1　　0|1=1　　1 | 1=1

例如，x=0000 0000 0000 1011，y=0000 0000 0000 1010，则表达式 x | y 的计算结果如下：

```
    0000 0000 0000 1011
(|) 0000 0000 0000 1010
    0000 0000 0000 1011
```

在【思考题 9-1】中，语句：

```
c2=a | b;        /*对 a 与 b 进行或运算，结果赋值给 c2*/
```

代码：c2=a | b;就是一个按位或的运算，当输入整数 a、b 的值分别为 12 和 23 时，系统将其自动转换为二进制数，代码 a | b 相当于 0000 1100 | 0001 0111，最后得到二进制编码为 0001 1111，转换成十进制数为 31。

按位或运算的应用：常用于将数据中的某些数位置 1，而其余数位保持不变。希望置 1 的数位与 1 进行“|”运算，保持不变的数位与 0 进行“|”运算。

4）按位异或运算

按位异或运算符用“∧”表示，规则是：若两个运算对象的对应二进制位不同，则结果的对应数位为 1，否则为 0。按位异或运算可能的运算组合及其运算结果如下所示：

0∧0=0　　1∧0=1　　0∧1=1　　1∧1=0

例如，x=0000 0000 0000 1011，y=0000 0000 0000 1010，则表达式 x∧y 的计算结果如下：

```
    0000 0000 0000 1011
(∧) 0000 0000 0000 1010
    0000 0000 0000 0001
```

在【思考题 9-1】中，语句：

```
c3=a∧b;        /*对 a 与 b 进行异或运算，结果赋值给 c3*/
```

代码：c3=a∧b;就是一个按位异或的运算，当输入整数 a、b 的值分别为 12 和 23 时，系统将其自动转换为二进制数，代码 a∧b 相当于 0000 1100∧0001 0111，最后得到二进制编码为 0001 1011，转换成十进制数为 27。

按位异或运算的应用如下。

（1）使数据的某些数位取反，即 1 变为 0，0 变为 1。要使数据的某位取反，只要使其与 1 进行“∧”运算；要使数据的某位保持原值，只要使其与 0 进行∧运算。

（2）取一个数据的某些数位。如原数为 0000 0000 0000 1011，若只保留第 2 位和第 3 位，只需将相应的位数设为 1，其余数位写 0，即与 0000 0000 0000 0110 进行∧运算即可。

（3）不用临时变量，交换两个变量的值。

5）左移运算符

左移运算符用“<<”表示，规则是：将运算对象中的每个二进制数位向左移动若干位，从左边移出去的高位部分被丢弃，右边空出的低位部分用“0”补齐。

例如：x=0000 0000 0000 1011，则 x<<2 的结果为 0000 0000 0010 1100。

在【思考题 9-1】中，语句：

```
c5=a<<2;        /*a 的各位全部左移 2 位，结果赋值给 c5*/
```

当输入 a 的值为 12 时，转换成二进制为 0000 1100，代码 a<<2 相当于 0000 1100<<2，得到二进制编码 0011 0000，转换成十进制数为 48。

在进行左移运算时，如果移出去的高位部分不包含数位 1，则每左移 1 位相当于该数乘以 2，左移 2 位相当于该数乘以 2^2=4，依次类推。因此在实际应用中，经常利用左移运算来进行乘 2 的操作。

6）右移运算符

右移运算符用“>>”表示，规则是：将运算对象中的每个二进制数位向右移动若干位，从右边移出去的高位部分被丢弃，左边空出的高位部分的处理分两种情况。对无符号数和正数来讲，左边空出的高位部分补“0”。对负数来讲，左边空出的高位部分补“0”还是补“1”，与所使用的编译程序有关，有的编译程序补“0”，称为逻辑右移；有的编译程序补“1”，称为算术右移。

例如：x=0000 0000 0000 1011，则 x>>2 的结果为 0000 0000 0000 0010。如果当 x 为负，如 x=1000 0000 0000 1011，则 x>>2 按逻辑右移的结果为 0010 0000 0000 0010，按算术右移的结果为 1110 0000 0000 0010。

在【思考题 9-1】中，语句：

```
c6=a>>2;        /*a 的各位全部右移 2 位，结果赋值给 c6*/
```

当输入 a 的值为 12 时，转换成二进制为 0000 1100，代码 a>>2 相当于 0000 1100>>2，得到二进制编码 0000 0011，转换成十进制数为 3。

二、位段

【思考题 9-2】 定义一个位域结构 bs，3 个位域为 a、b、c，说明 bs 类型的变量 bit 和指向 bs 类型的指针变量 pbit。分别给 3 个位域赋值 1、7、15，并以整型量格式输出 3 个域的内容。把位域变量 bit 的地址送给指针变量 pbit，用指针方式给位域 a 重新赋值，赋为 0。参考下面的源程序代码，了解位段的基本知识。



程序分析

在这里主要使用了位段的定义。要了解位段，就必须知道位段如何定义、如何进行引用。



编写程序代码

```
main()
{   struct  bs
    {  unsigned int a:1;
       unsigned int b:3;
       unsigned int c:4;
    }bit,*pbit;
    bit.a=1;
    bit.b=7;
    bit.c=15;
    printf("%d,%d,%d\n",bit.a,bit.b,bit.c);
}
```



调试运行程序

程序运行结果如下：

```
1,7,15
```



3．位段的定义

在程序设计中，有时存储一个信息不必用一个或多个字节，可以在一个字节中存放一个

或多个信息。例如，“真”或“假”用 1 或 0 表示，只需一位即可，如果用一个变量来存储，则将浪费存储空间。为了解决该问题，C 语言提供了位段操作。

所谓位段，是由一个或多个二进制数位组成的，它是数据的一种压缩形式。位段是一种特殊的压缩形式结构体结构中的成员，它的特殊性在于它是以位为单位定义长度的。

位段的定义格式如下：

```
类型标识符    成员名：二进制位数;
```

其中，位段成员的类型一般为 unsigned 型，二进制位数指明了该位段的长度，以二进制为单位。例如【思考题 9-2】中：

```
struct  bs
   {  unsigned  int  a:1;
      unsigned  int  b:3;
      unsigned  int  c:4;
   }bit,*pbit;
```

定义了结构体 bs，该结构体变量共有 3 个成员，其中 a、b、c 成员是位段，分别占 1 位、3 位、4 位，共占 1 字节。



4．位段的引用

位段的引用方法与引用结构体变量中的成员相同，即

```
结构体变量名.成员名
```

例如在【思考题 9-2】中，可以通过赋值语句给位段赋值，如：

```
bit.a=1;
bit.b=7;
bit.c=15;
```



在对位段赋值时，应该考虑到每个位段所占用的二进制位数，如果所赋值的数值超过了位段的表示范围，则自动取其低位数字。

例如，下面的赋值语句就有问题：

```
bit.b=10;
```

因为 bit.b 占 3 位，最大值为 $2^3-1=7$，而给它赋值为 10，将会出现溢出，在此情况下，系统自动取赋予它的值 10 的二进制数的低 3 位（010）。

在定义和使用位段时，应该注意的问题如下。

（1）在位段结构中可以定义无名位段，它起位段分隔的作用。例如：

```
struct  bs
{  unsigned  int  a:1;
   unsigned  int  b:3;
   unsigned  int   :8    /*这 8 位空间不用*/
   unsigned  int  c:4;
} bit;
```

在这个结构体变量 bit 中，有 4 个位段成员，其中第 3 个位段为无名位段，它占 8 位，在 b 和 c 位段之间起分隔作用，即在 b 成员后面空出 8 位不用，接着是 c 成员占 4 位。

（2）若某一位段要从另一个存储单元开始存放，可以使用长度为 0 的位段来定义。例如：

```
struct  bs
{  unsigned  int  a:1;
   unsigned  int  b:3;
   unsigned  int   :0;
   unsigned  int  c:4;
} bit;
```

在结构体变量 bit 中，有 4 个位段成员，其中一个长度为 0 的无名字段，它使得 a 和 b 成员放在一个存储单元中，而成员 c 放在另一个存储单元中。

（3）位段一般用 unsigned int 型，并且在位段结构体定义中，可以包含非位段成员。例如：

```
struct  bs
{  unsigned  int  a:1;
   unsigned  int  b:3;
   unsigned  int  c:4;
   int i;
 } bit;
```

其中，非位段成员 i 在 c 成员之后从另一个字节开始存放。

（4）一个位段必须存储在同一个存储单元中，不能跨两个存储单元。因此，每个位段所占用的二进制位数通常不能超过一个字长。

（5）位段可以用整型格式符输出，也可以在一般表达式中被引用，并自动转换为相应的整数。例如【思考题 9-2】改成如下形式：

```
main()
{  int i;
   struct bs
   {  unsigned a:1;
      unsigned b:3;
      unsigned c:4;
    } bit;
    bit.a=1;
    bit.b=7;
    bit.c=15;
    printf("%d,%d,%d\n",bit.a,bit.b,bit.c);
    i=bit.a+bit.b+bit.c ;
    printf("%d,%o,%x,%d\n",bit.a,bit.b,bit.c,i);
}
```

程序运行结果如下：

```
1,7,15
1,7,f,23
```

练一练

【练习 9-1】 从键盘上输入一个正整数 n，判断此数是奇数还是偶数。

解：奇数的二进制表示中右边的第 1 位为 1，偶数的二进制表示中右边的第 1 位为 0。因此该题就转换为取该数右边的第 1 位，并判断其值是否为 0。源程序如下：

```
main()
{  int  n;
   printf("input n(n>0):");
   scanf("%d",&n);
```

```
    if((n&0x01)==0)  /*取 n 的最后一位，若其值为 0，则为偶数，否则为奇数*/
      printf("%d is a even number.\n",n);
    else
      printf("%d is a odd number.\n",n);
  }
```

程序运行结果如下：

```
input n(n>0):10
10 is a even number.
input n(n>0):9
9 is a odd number.
```

【练习 9-2】 从键盘上输入一个字符，统计该字符的 ASCII 码中 1 的个数，并显示出来。

解：对输入的字符的 ASCII 码的每一位进行测试，视其为 1 还是为 0 进行相关运算。可以设置一个屏蔽字与该数进行“&”运算，从而取出所需的某位的状态。从最高位开始，第 8 位所需设置的屏蔽字 mask 为 0x80，即二进制数 1000 0000，若 ch&mask 的结果为 0，则说明 ch 的第 8 位为 0，否则为 1。第 7 位所需设置的屏蔽字 mask 为 0x40，即将 mask 右移一位。源程序代码如下：

```
#include  "stdio.h"
main()
{  char  ch;
   int  mask,i,count=0;
   printf("\ninput   ch:");
   ch=getchar();
   mask=0x80;
   for(i=0;i<8;i++)
   {  if((ch&mask)!=0)   count++;
      mask>>=1;
   }
   printf("ASCII is  %x  ,count=%d\n",ch,count);
}
```

程序运行结果如下：

```
input   ch:x
ASCII is  78  ,count=4
```

【练习 9-3】 将十六进制数转换为二进制数。

解：人们有时希望知道某个十六进制数的二进制数是什么，但 C 语言的 printf 函数只提供%x，%d，%o 方式输出一个整数（即十六进制、十进制、八进制）而不能直接输出一个整数的二进制形式，需要人工转换，不方便。在这里可以用位运算来实现此功能。

采用的方法是：对一个整数 num（16 位）的每一位进行测试，视其为 0 还是为 1，可以设置一个屏蔽字与该数进行 & 运算，从而保留（取出）所需的一个位的状态。从高低（第 15 位）开始，此时设置的屏蔽字为 0x8000（十六进制数 8000），即二进制数 1000 0000 0000 0000，其最高位（第 15 位）为 1，其余位为 0，将它赋予一个变量 mask，使 mask & num，若结果为 1 说明 num 的第 15 位为 1，否则为 0。把这个 1 或 0 存放在 bit 中，即

```
bit=(mask & num)?1:0;
```

这个 bit 的值（非 1 即 0）就是 num 第 15 位之值。下面处理第 14 位，此时 mask 应改为 0x4000，也就是使 mask 右移一位。源程序如下：

```
main()
{  int  j,num,bit;
```

```
    unsigned  int mask;
    mask=0x8000;
    printf("\nenter your number:");
    scanf("%x", &num);
    printf("binary of % 04x is :",num);
    for(j=0;j<16;j++)
    {  bit=(mask & num)?1:0;
       printf("%d",bit);
    if(j==7)
      printf("--");
    mask>>=1;
    }
}
```



输入时用十六进制形式输入数据，程序输出相应的二进制形式。

程序运行结果如下：

```
enter your number:1
binary of 0001 is :00000000--00000001
enter your number:f1e2
binary of f1e2 is :11110001--11100010
enter your number:cdef
binary of cdef is :11001101--11101111
```

想一想

位运算都有哪些功能，在什么情况下我们可以用到位运算？

本章小结

由于 C 语言最初是为了描述系统程序而设计的，因此与大多数高级语言不同，C 语言提供了位运算的功能。有了位运算，C 语言就能编写出直接对计算机硬件进行操作的程序，具有汇编语言的一些功能。

所谓位运算，是指对二进制数位进行的运算。每一个二进制数位只能存放 0 或 1，因此位运算符的运算对象是一个二进制数位的集合。C 语言提供了六种位运算符：按位与、按位或、按位异或、按位取反、左移和右移。

在程序设计中，有时存储一个信息不必用一个或多个字节，可以在一个字节中存放一个或多个信息。例如，“真”或“假”用 1 或 0 表示，只需一位即可，如果用一个变量来存储，则会浪费存储空间。为了解决该问题，C 语言提供了位段操作。所谓位段是由一个或多个二进制数位组成的，它是数据的一种压缩形式。

课后习题九

一、选择题

1．在 C 语言中，要求操作数必须是整型或字符型的运算符是_____。

A）&&　　B）&　　C）！　　D）||

2．表达式 a<b||～c&d 的运算顺序是_____。

A）～，&，<，||　　B）～，||，&，<

C）～，&，‖，<　　D）～，<，&，‖

3．以下叙述不正确的是_____。

A）表达式 a&=b 等价于 a=a&b　　B）表达式 a|=b 等价于 a=a|b

C）表达式 a!=b 等价于 a=a!b　　D）表达式 a∧=b 等价于 a=a∧b

4．在执行完以下 C 语句后，B 的值是_____。

```
char  z='A';
int   b;
b=((241&15)&&(z|'a'));
```

A）0　　B）1　　C）TRUE　　D）FALSE

5．表达式 0x13∧0x17 的值是_____。

A）0x04　　B）0x13　　C）0xE8　　D）0x17

6．请读以下程序片断，输出结果是_____。

```
int  x=20;
printf("%d",～x);
```

A）02　　B）-20　　C）-21　　D）-11

7．表达式～0x13 的值是_____。

A）0xffec　　B）0xff71　　C）0xff68　　D）0xff17

8．设有以下语句：

```
char  x=3,y=6,z;
z=x∧y<<2;
```

则 z 的二进制值是_____。

A）0001 0100　　B）0001 1011　　C）0001 1100　　D）0001 1000

二、填空题

1. 设 x=1010 0011，若要通过 x∧y 使 x 的低 4 位取反，高 4 位不变，则 y 表示的二进制数为_____。

2．下面语句执行后，y 表示的二进制数为_____。

```
unsigned  y;
y=～(～0<<4);
```

3．下面语句执行后，z 表示的二进制数为_____。

```
unsigned  a=0112,x,y,z;
x=a>>3;
y=～(～0<<4);
z=x&y;
```

4．若有以下语句，则 c 表示的二进制数为_____。

```
short  int  a=3,b=6,c;
c=a∧b>>2;
```

5．设有定义 char a,b;，若想通过 a&b 保留 a 的第 3 位和第 6 位的值，则 b 表示的二进制数应是_____。

三、编程题

1．输入两个正数存入 a 和 b 中，并由 a 和 b 两个数生成新的数 c，其生成规则是：将 a 的低字节作为 c 的高字节，将 b 的低字节作为 c 的低字节，并显示出来。

2．设计一个程序，实现循环左移操作。循环左移是指将左端移出的数位补到右端。例如，将1100 0011 0000 0000 循环左移 3 位，变为 0001 1000 0000 0110。

3．取一个整数 a 从右端开始的 4～7 位。

第十章　文　　件

在程序运行时，程序本身和数据一般都存放在内存中，当程序运行结束后，存放在内存中的数据被释放。如果需要长期保存程序运行所需的原始数据或程序运行产生的结果，就必须以文件形式存储到外部存储介质上。

为了提高数据的处理效率，一般高级语言都能对文件进行操作。文件可以是自己编制的，也可以是系统已有的。无论是程序或数据，都是以文件方式存储的。本章主要介绍文件的一般概念，文件指针以及文件的打开、关闭、读、写、定位等操作。

第二十五讲　文件概述、文件打开与关闭



学习目标

- 掌握文件的基本概念;
- 掌握文件的打开方法;
- 掌握文件的关闭方法。

【思考题 10-1】 如果我们需要将程序运行结果存放到外存时，应该用什么方法存储呢？应该怎样读入数据或存储数据呢？



程序分析

我们如果要将一个字符串"This is a C program."存放到一个文件中，应该怎么设计程序呢？请看下例。



编写程序代码

```
#include  <stdio.h>
main()
{  FILE  *fp;
   fp=fopen("file1.txt","w");
   fprintf(fp,"This is a C program.");
   fclose(fp);
}
```



调试运行程序

在这个程序中，建立了一个名为"file1.txt"的文件，并将字符串"This is a C program."通过 fprintf 函数输出到文件中保存起来。同理，也可以用 fscanf 函数从文件"file1.txt"中输入该内容。该程序在屏幕上没有输出信息。下面我们首先来介绍一下文件的相关概念。

小讲堂

1．文件的定义

所谓文件，是指存储在外部介质上的数据的集合。计算机把所有外部设备都当做文件来对待，这样的文件称为设备文件。例如，键盘是输入文件，显示屏和打印机是输出文件，我们可以用 scanf 函数输入键盘文件，printf 函数输出显示屏和打印机文件。实际上，外部设备的输入/输出操作就是读/写设备文件的过程，对设备文件的读/写与对一般磁盘文件的读/写方法完全相同。

小讲堂

2．数据文件的存储形式

C 语言把文件看做是一个字符（字节）的序列，即由一个个字符（字节）的数据顺序组成。根据数据的组织形式，可分为字符文件和二进制文件。

1）字符文件（文本文件）

字符文件又称文本（text）文件，它的每一个字节放一个 ASCII 代码，代表一个字符（即一个字符占一字节）。

2）二进制文件

二进制是把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放，这类文件可以节省内存空间。

如有一个整数为 2358，在内存中的存放形式为 0000 1001 0011 0110，采用二进制文件形式占 2 字节。如果用字符文件形式存放 2358 到磁盘上时，由于字符的 ASCII 码与字符一一对应，一个字节代表一个字符，即需要存放'2'、'3'、'5'、'8'四个字符，因此需占 4 字节。

虽然用 ASCII 形式存放比按二进制形式存放占较多的存储空间，但是这便于对字符进行逐个处理和输出。

一个 C 文件是一个字节流或二进制流。它把数据看做是一连串的字符（字节），而不考虑记录的界限。也就是说，C 语言文件不是由记录组成的。在 C 语言中对文件的存取是以字符（字节）为单位的，输入/输出的数据流的开始和结束仅受程序控制而不受物理符号（如回车换行符）控制，一般把这种文件称为流式文件。C 语言允许对文件存取一个字符，这就增加了处理的灵活性。

小讲堂

3．标准文件系统和非标准文件系统

在过去使用的 C 版本中（如 UNIX 系统下使用的 C）有两种对文件的处理方法：一种叫“缓冲文件系统”，一种叫“非缓冲文件系统”。

1）缓冲文件系统

所谓缓冲文件系统，是指系统自动地在内存区为每一个正在使用的文件名开辟一个缓冲区。从内存向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才一起送到磁盘去。如果从磁盘向内存读入数据，则一次从磁盘文件中将一批数据读入到内存缓冲区（充满缓冲区），然后再从缓冲区逐个地将数据送到程序数据区（给程序变量），如图 10-1 所示。缓冲

区的大小由各个具体的 C 版本确定，一般为 512 字节。

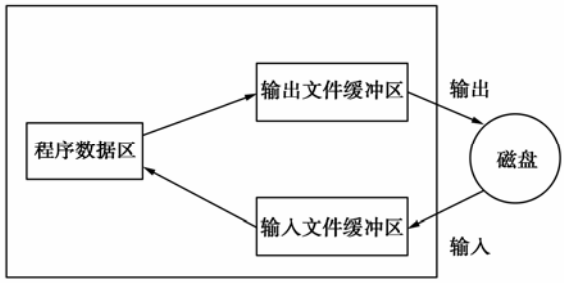


图 10-1　从磁盘文件向内存读入数据和内存输出数据

2）非缓冲文件系统

所谓“非缓冲文件系统”是指系统不自动开辟确定大小的缓冲区，而由程序为每个文件设定缓冲区。在 UNIX 系统下，用缓冲文件系统来处理文本文件，用非缓冲文件系统来处理二进制文件。用缓冲文件系统进行的输入/输出又称为高级（或高层）磁盘输入/输出，用非缓冲文件系统进行的输入/输出又称为低级（或低层）磁盘输入/输出。1983 年 ANSI C 标准规定不采用非缓冲文件系统，而只采用缓冲文件系统。也就是说，用缓冲文件系统既可以处理二进制文件，又可以处理文本文件。这里主要讨论 ANSI C 的文件系统以及对它的读写操作。



4．文件类型指针

1）文件类型 FILE 的定义

在进行文件操作时，要用到文件指针。文件指针用来指向被操作文件的有关信息（如文件名、文件状态及文件当前位置等）。这些信息是保存在一个结构体变量中的。该结构体类型是由系统定义的，类型名为 FILE。有的 C 版本在 stdio.h 文件中有以下的文件类型声明：

```
typedef  struct
{  int  fd;                    /*文件号*/
   int  cleft;                 /*缓冲区中剩下的字符*/
   int  mode;                  /*文件操作模式*/
   char *nextc;                /*下一个字符位置*/
   char *buffer;               /*文件缓冲区位置*/
}FILE;
```

有了 FILE 类型以后可以定义文件类型的指针变量。



FILE 是一个标准的标识符（即文件类型名），但不是关键字，它是被定义出来的结构体变量类型名。

2）文件类型指针的定义

（1）文件类型指针的定义格式如下：

```
FILE    *指针变量标识符;
```

（2）功能。定义一个文件类型的指针，使用这个指针可以对文件进行各种操作。

例如在【思考题 10-1】程序中就有下列语句：

```
FILE  *fp;
```

其中，fp 是一个指向 FILE 类型结构体的指针变量。可以使 fp 指向某一个文件的结构体变量，从而能够通过该结构体变量中的文件信息去访问该文件。也就是说，通过文件指针变量 fp 能够找到与之相关的文件“file1.txt”。一般来说，打开多少个文件，就应该有多少个文件型指针变量，使它们指向对应的文件（实际上是指向存放该文件信息的结构体变量），以实现对文件的访问。

3）标准文件操作的四个基本步骤

（1）文件类型指针的定义；

（2）打开标准文件；

（3）标准文件的读或写的操作；

（4）标准文件的关闭操作。



在对文件操作之前必须“打开”文件，打开文件的作用实际上是建立该文件的信息结构体，并且给出指向该信息结构体的指针以便对该文件进行访问。文件使用结束之后应该“关闭”该文件。文件的打开与关闭是通过调用 fopen 和 fclose 函数来实现的。

5．文件的打开（fopen函数）

ANSI C 规定了标准输入/输出函数库，用 fopen()函数来实现文件的打开，其调用的一般格式如下：

```
FILE   *fp;
fp=fopen(fname,mode) ;
```

其中，fname 是要打开的文件名，可以是字符串常数、字符型数组或字符型指针，文件名也可以带路径。mode 表示文件的使用方式，它规定了打开文件的目的，如表 10-1 所示。

表 10-1　fopen 函数中的文件使用方式

文件使用方式	含　义	说　明
"r"（只读）	打开文本文件，只读	如果指定文件不存在，则出错
"w"（只写）	打开文本文件，只写	新建一个文件，如果指定文件已存在，则删除它，再新建
"a"（追加）	打开文本文件，追加	如果指定文件不存在，则创建该文件
"rb"（只读）	打开二进制文件，只读	如果指定文件不存在，则出错
"wb"（只写）	打开二进制文件，只写	新建一个文件，如果指定文件已存在，则删除它，再新建
"ab"（追加）	打开二进制文件，追加	如果指定文件不存在，则创建该文件
"r+"（读写）	打开文本文件，读、写	如果指定文件不存在，则出错
"w+"（读写）	打开文本文件，读、写	新建一个文件，如果指定文件已存在，则删除它，再新建
"a+"（读追加）	打开文本文件，读、追加	如果指定文件不存在，则创建该文件
"rb+"（读写）	打开二进制文件，读、写	如果指定文件不存在，则出错
"wb+"（读写）	打开二进制文件，读、写	新建一个文件，如果指定文件已存在，则删除它，再新建
"ab+"（读追加）	打开二进制文件，读、追加	如果指定文件不存在，则创建该文件

注：各字符含义如下：r:（read）, w:（write）, a:（append）, b:（binary）, +:（读写）。

例如，有如下语句：

```
FILE *fp;
fp=fopen("file1.txt","w");
```

它表示要打开名字为"file1.txt"的文件，使用文件方式为“写入”，fopen()函数返回值是指向文件"file1.txt"的指针，将其赋给 fp，这样 fp 就指向了文件"file1.txt"。



如果文件名中包括文件的路径，则用双反斜线表示路径（双反斜线“\\”中的第一个表示转义字符，第二个表示路径分隔符“\”）。例如：

```
FILE *fp;
fp=fopen("c:\\user\\abc.txt","w");
```

其意义是以只写方式打开 C 驱动器磁盘下文件夹 user 中的文件 abc.txt，并使文件指针 fp 指向该文件。

说明：

（1）用以上方式可以打开文本文件或二进制文件，这是 ANSI C 的规定，即用同一种文件缓冲系统来处理文本文件和二进制文件。但目前有些 C 编译系统可能不完全提供所有这些功能，有的 C 版本不用"r+"、"w+"、"a+"而用"rw"、"wr"、"ar"等。请大家注意所用 C 系统的规定。

（2）如果文件“打开”不能实现，fopen()函数值将会返回一个错误信息。出错的原因可能是：用"r"方式打开一个并不存在的文件，磁盘出故障，磁盘已满无法建立新文件等。此时 fopen()函数将返回一个空指针值 NULL（NULL 在 stdio.h 文件中已被定义为 0）。

常用下列方法打开一个文件：

```
if((fp=fopen("file1","r"))==NULL)
{  printf("cannot  open  this  file \n");
   exit(0);
}
```

即先检查打开文件（file1）有无出错，如果有错就在终端上输出“cannot open this file”。exit（0）函数的作用是关闭所有文件，终止正在调用的进程。

（3）在读取文本文件时，会自动将回车、换行两个符号转换为一个换行符；在写入时会自动将一个换行符转换为回车和换行两个字符。在用二进制文件时，不会进行这种转换，因为在内存中的数据形式与写入到外部文件中的数据形式完全一致，一一对应。

（4）在程序开始运行时，系统自动打开三个文件：标准输入、标准输出、标准出错输出，通常这三个文件都与终端相联系。因此以前我们所用到的从终端输入或输出，都不需要打开终端文件。系统自动定义了三个文件指针 stdin、stdout 和 stderr，分别指向终端输入、终端输出和标准出错输出（也从终端输出）。如果程序中指定要从 stdin 所指的文件输入数据，就是指从终端键盘输入数据。



6．文件的关闭（fclose 函数）

文件使用完后应将它关闭，以保证本次文件操作的有效。“关闭”就是使文件指针变量不指向该文件，也就是文件指针变量与文件“脱钩”。此后不能再通过该指针对原来关联的文件进行操作。

用 fclose 函数关闭文件，其一般形式为：

fclose(文件指针);

例如：

```
fclose(fp);
```

在前面例子中，我们把 fopen 函数带回的指针赋给了 fp，现在通过 fp 关闭该文件，即 fp 不再指向该文件。

在程序终止之前应关闭所有使用的文件，否则将会丢失数据。这是因为在向文件写数据时，是先将数据写入缓冲区，等缓冲区写满后才真正输出给文件。如果缓冲区未满而程序结束运行，就会将缓冲区中的数据丢失。用 fclose 函数关闭文件，可以避免这一问题的发生。

如果文件关闭成功，fclose 函数返回值为 0，否则返回 EOF(−1)。这可以用 ferror 函数来测试。

练一练

【练习 10-1】 以只读的方式打开 D 盘的"d:\abc.txt"文件，语句应该怎么写？

解：打开文件函数为 fopen，第一个参数为文件名；第二个为打开文件的方式，只读方式的模式为"r"，所以语句写为：

```
FILE  *fp;
fp=fopen("d:\\abc.txt","r");
```

【练习 10-2】 以读写的方式打开 D 盘的"d:\abc.txt"文件，语句应该怎么写？（注意假设 D 盘中有该文件和没有该文件的两种情况。）

解：打开文件函数为 fopen，第一个参数为文件名，文件名注意两侧有双括号引上，并将单个的斜线“\”改为双斜线“\\”即可。第二个为打开文件的方式，读写方式的方式为"r+"或"w+"。使用"r+"的情况是在不删除原有文件时（没有该文件时会出错）；而"w+"则会删除原有文件，然后新建一个该名的文件。所以语句写为：

```
FILE  *fp;
fp=fopen("d:\\abc.txt ","r+");  /*不删除原有文件时*/
```

或：

```
fp=fopen("d:\\abc.txt ","w+");  /*删除后新建该文件*/
```

【练习 10-3】 在【练习 10-1】中打开"d:\abc.txt"文件后，如果想关闭该文件的语句是什么？

解：关闭文件的函数为 fclose，参数为要关闭的文件指针名。所以语句写为：

```
fclose(fp);
```

本讲小结

在本讲中主要介绍了文件的概念，数据文件的存储形式，什么是标准文件系统和非标准文件系统，文件类型指针 FILE，文件的打开（fopen 函数）和文件的关闭（fclose 函数）。要注意文件的打开函数的各种方式的使用方法。

想一想

在打开某一文件后，如果用户想对文件中的信息进行操作（如读取数据或存储数据），应该如何进行呢？

第二十六讲　文件读写

学习目标

- 掌握 fputc 函数和 fgetc 函数;
- 掌握 fputs 函数和 fgets 函数;
- 掌握 fwrite 函数和 fread 函数;
- 掌握 fprintf 函数和 fscanf 函数。

一、fputc函数和fgetc函数

【思考题 10-2】 若想将一些字符写入一个文本文件，然后输出到屏幕上，应该如何读写程序？

程序分析

设计一个程序，将字符 A、B、C 和 EOF 写入文件"c:\file1.txt"中，然后再从文件"c:\file1.txt"中读出所有的字符并显示在屏幕上。

编写程序代码

```
#include "stdio.h"
main()
{  char  a='A',b='B',c='C',filename[20];
   FILE  *fp;
   clrscr();                                  /*清屏*/
   printf("\nPlease input the write filename:");
   scanf("%s",filename);                      /*从键盘读入文件名*/
   if((fp=fopen(filename,"w"))==NULL)    /*以只写方式打开文件*/
   {  printf("write cannot open file\n");
      exit(0);
   }
   fputc(a,fp);                               /*将字符变量 a 的值存入文件*/
   fputc(b,fp);                               /*将字符变量 b 的值存入文件*/
   fputc(c,fp);                               /*将字符变量 c 的值存入文件*/
   fputc(0xff,fp);                            /*将文件结束标志 EOF 存入文件*/
   fclose(fp);                                /*关闭该文件*/
   /*以下为输出文件内容*/
   if((fp=fopen(filename,"r"))==NULL)    /*以只读方式打开该文件*/
   {  printf("read cannot open file\n");
      exit(0);
   }
   while((a=fgetc(fp))!=EOF)                  /*当文件指针没到文件尾部时*/
      putchar(a);
   fclose(fp);                                /*关闭该文件*/
}
```

调试运行程序

程序运行时，用户先输入文件路径及文件名"c:\file1.txt"，程序以"w"方式打开文件，然后由函数 fputc()依次将保存在变量 a、b、c 中的字符写到文件中，最后写入 EOF 并关闭文件。通过 Windows 资源管理器可以看到文件 file1.txt 被建立，其长度为 4 字节。然后程序再循环

从该文件中输出 A、B、C 三个字符并显示到屏幕上，当遇到文件结束标志时循环结束。

程序运行结果如下：

```
Please input the write filename:c:\file1.txt
ABC_
```

小讲堂

1．fputc函数

fputc 函数是把一个字符输出（写入）到磁盘文件上，与其完全等价的还有 putc()。它们的一般调用形式为：

`fputc(ch, fp);` 和 `putc(ch,fp);`

例如，【思考题 10-2】程序中的语句：

```
fputc(a,fp);                    /*将字符变量 a 的值存入文件*/
fputc(b,fp);                    /*将字符变量 b 的值存入文件*/
fputc(c,fp);                    /*将字符变量 c 的值存入文件*/
fputc(0xff,fp);                 /*将文件结束标志 EOF 存入文件*/
```

就是将三个字符变量和一个文件结束标志字符写入文件中。

其中，ch 是要输出的字符，它可以是一个字符常量，也可以是一个字符变量。fp 是文件指针变量。fputc(ch,fp)函数的作用是将字符（ch 的值）输出到 fp 所指向的文件上。如果输出成功，函数的返回值是输出的字符；如果输出失败，则返回文件结束标志 EOF。EOF 是在 stdio.h 中定义的符号常量，值为−1，十六进制表示为 0xFF。

文本文件也可以用 DOS 的 TYPE 命令将其内容显示在屏幕上（EOF 将按照 ASCII 码值为 255 显示）。

小讲堂

2．fgetc函数

fgetc 函数是用来从指定文件中读取一个字符，与它完全等价的还有 getc()。它们的调用格式如下：

`ch=fgetc(fp);` 和 `ch=getc(fp);`

其中，fp 为文件指针。该函数的功能是从指定的文件读取一个字符，并赋给字符型变量 ch。如果读取成功，函数返回读取的字符；如果遇到文件结束符，则返回一个文件结束标志 EOF。

二、fputs函数和fgets函数

【思考题 10-3】 若想将多个字符串写入一个文本文件，然后输出到屏幕上，应该如何读写程序？

程序分析

设计一个程序，将字符串"Hello,"、"all "、"the "、"world "、"people!"写入文件"c:\file2.txt"

中，然后再从文件"c:\file2.txt"中读出所有的字符串并显示在屏幕上。



编写程序代码

```
#include  "stdio.h"
main()
{ char  a[][9]={"Hello,","all ","the ","world ","people!"};
  char  filename[20],out[9];
  int  i;
  FILE  *fp;
  clrscr();                                    /*清屏*/
  printf("\nPlease input the write filename:");
  scanf("%s",filename);                        /*从键盘读入文件名*/
  if((fp=fopen(filename,"w"))==NULL)           /*以只写方式打开文件*/
  {  printf("write cannot open file\n");
     exit(0);
  }
  for(i=0;i<=4;i++)
     fputs(a[i],fp);                           /*将字符串存入文件*/
  fclose(fp);
  /*以下为从文件中读出各字符串并显示在屏幕上*/
  if((fp=fopen(filename,"r"))==NULL)           /*以只读方式打开该文件*/
  {  printf("read cannot open file\n");
     exit(0);
  }
  while((fgets(out,9,fp))!=NULL)   /*当 fp 指针没到文件尾部，取出一字符串*/
     printf("%s",out);
  fclose(fp);                                  /*关闭该文件*/
}
```



调试运行程序

程序运行时，用户先输入文件路径及文件名"c:\file2.txt"，程序以"w"方式打开文件"c:\file2.txt"，然后由 fputs()依次将字符数组 a 中的各字符串写到文件"c:\file2.txt"中。然后程序再用循环方式从该文件中读出各字符串并显示到屏幕上。

程序运行结果如下：

```
Please input the write filename:c:\file2.txt
Hello,all the world people!_
```



3．写文件字符串函数fputs()

（1）函数调用格式如下：

```
fputs(字符串，文件指针);
```

其中，字符串可以是一个字符串常量，或字符数组名，或字符指针变量名。

（2）功能。向指定文件输出一个字符串，同时将读写位置指针向前移动 strlength（字符串长度）字节。如果输出成功，则函数返回值为 0；否则，为非 0 值。例如，在例程中语句：

```
fputs(a[i],fp);                    /*将字符串存入文件*/
```

就是将二维字符数组 a 中的每一行字符串存入到 fp 所指文件中。



4．读文件字符串函数fgets()

（1）函数调用格式如下：

fgets(字符数组名/指针名，字符串长度+1，文件指针);

（2）功能。从指定文件中读入一个字符串，存入字符数组/指针中，并在尾端自动加一个结束标志'\0'；同时，将读写位置指针向前移动 strlength（字符串长度+1）字节。如果在读入规定长度之前遇到文件尾 EOF 或换行符，读入即结束。例如，在例程中的语句：

```
while((fgets(out,9,fp))!=NULL)     /*当文件指针没到文件尾部时*/
```

该语句中的 fgets(out,9,fp)就表示将从文件中读出的字符串存入到字符数组 out 中，且每次字符串移动位置是 9 个字符。

三、fwrite函数和fread函数

【思考题 10-4】 若想向文件中输入或从文件中读出结构体类型（成员为多种数据类型）的信息，应该用什么样的方法比较好？



程序分析

设计一个程序，从键盘输入一批学生数据，然后存储到磁盘文件上，再输出磁盘文件中的学生数据到屏幕上。注意成块输入 fread 函数和成块输出 fwrite 函数的使用方法。



编写程序代码

```
#include  "stdio.h"
#include  "string.h"
struct  student                          /*定义学生结构体类型*/
{  int  num;                             /*定义学号*/
   char  name[20];                       /*定义姓名*/
   int  chinese,math,english;            /*定义语文，数学，英语成绩*/
};
main()
{  FILE  *fp;
   int  i;
   char  ch,filename[30];
   float  f;
   struct  student  stu[30],st;          /*定义结构体类型的变量*/
   clrscr();                             /*清屏*/
   printf("\nPlease input the write filename:");
   scanf("%s",filename);                 /*从键盘读入文件名*/
   if((fp=fopen(filename,"wb"))==NULL)   /*以只写方式打开文件*/
   {  printf("Cannot open %s file\n",filename);
      exit(0);
   }
   /*循环读入一批学生信息，当输入 n 或 N 时输入结束输入*/
   i=0;
```

```
do
{  printf("please input information of the  %d student:",i+1);
   printf("\nnumber  name  chinese  math  english\n");
   scanf("%d%s%d%d%d",&stu[i].num,stu[i].name,&stu[i].chinese,
          &stu[i].math,&stu[i].english);
   fwrite(&stu[i],sizeof(stu[i]),1,fp);
   i++;
   printf("Have another student record(y/n)?");
   getchar();
   ch=getchar();
}while(ch=='y'||ch=='Y');              /*当输入 y 或 Y 时继续输入*/
fclose(fp);
/*以下功能为将学生成绩读出并显示到屏幕上*/
if((fp=fopen(filename,"rb"))==NULL)   /*以只读方式打开该文件*/
{  printf("read cannot open file %s!\n",filename);
   exit(0);
 }
 printf("\nInformation of the  student score:");
 printf("\n number  name  chinese   math   english");
 while(fread(&st,sizeof(st),1,fp)==1)
 printf("\n%5d%7s%5d%5d%5d",st.num,st.name,st.chinese,st.math,
        st.english);
 if(!feof(fp))
    printf("\n file read error");
 fclose(fp);
}
```



调试运行程序

首先输入文件名"c:\file3.txt"，程序以"wb"方式打开文件"c:\file3.txt"，然后输入第一个学生信息，由函数 fwrite()依次将一个学生信息写到文件"c:\file3.txt"中，然后出现提示是否继续输入下一条学生信息，当输入 y 或 Y 时继续输入，当输入 n 或 N 时输入结束输入，关闭该文件。然后以只读方式"rb"打开该文件，程序再用循环方式用函数 fread()从该文件中读出各学生信息并显示到屏幕上。

程序运行结果如下：

```
Please input the write filename:c:\file3.txt
please input information of the  1 student:
number  name  chinese  math  english
1 liu 89 66 94
Have another student record(y/n)?y
please input information of the  2 student:
number  name  chinese  math  english
2 wang 92 85 88
Have another student record(y/n)?y
please input information of the  3 student:
number  name  chinese  math  english
3 zhao 87 92 90
Have another student record(y/n)?n

Information of the  student score:
 number   name  chinese   math   english
    1     liu      89      66      94
    2    wang      92      85      88
    3    zhao      87      92      90_
```



fwrite 函数和 fread 函数是文件的数据块读/写函数，fwrite 函数和 fread 函数一般用于二进制文件的输入和输出。

5．fwrite函数

（1）fwrite 函数调用格式如下：

```
fwrite(buf,size,count,fp);
```

（2）说明。fwrite 函数用来向指定文件中写入数据块。其中，buf 为被写入数据在内存中存放的起始地址，可以是数组名或指向数组的指针；size 为每次要写入的字节数；count 为要写入的次数；fp 为文件指针。

（3）功能。该函数的功能是从 buf 所指向的内存区域取出 count 个数据项写入 fp 指向的文件中，每个数据项的长度为 size，也就是写入的数据块大小为 size*count 个字节。如果函数执行成功，返回值为实际写入数据项的个数，否则若返回值小于实际需要写入数据项的个数 count，则出错。当文件按二进制形式打开时，fwrite 函数可以写入任何类型的信息。



6．fread函数

（1）fread 函数调用格式如下：

```
fread(buf,size,count,fp);
```

（2）功能。该函数用于从文件中读取一个数据块。其中，buf 为从文件中读取的数据在内存中存放的起始地址，size 为一次读取的字节数，count 为要读取的次数，fp 为文件指针。该函数的功能是从 fp 所指的文件中，读取长度为 size 字节的数据项 count 次，存放到 buf 所指的内存单元中，所读取的数据块大小为 size*count 字节。当文件按二进制形式打开时，fread 函数可以读出任何类型的信息。函数执行成功时，返回值为实际读出的数据项个数，否则若返回值小于实际需要读出数据项的个数 count，则出错。

四、fprintf函数和fscanf函数

【思考题 10-5】 若想将多个不同类型的数据写入一个文本文件，然后输出到屏幕上，应该如何读写程序？



程序分析

设计一个程序，从键盘输入一批学生信息，并将学生信息写入到一个二进制文件中，然后存储到磁盘文件上，再输出磁盘文件中的学生信息到屏幕上。注意格式化输入 fprintf 函数和格式化输出 fscanf 函数的使用方法。



编写程序代码

```
#include  "stdio.h"
struct  student
{  char  name[10];
```

```
    int  num;
    int  age;
    char  addr[15];
}boy[2],*bp;
main()
{  FILE *fp;
   char ch,filename[20];
   int i;
   clrscr();                          /*清屏*/
   printf("\nPlease input the write filename:");
   gets(filename);                    /*读取文件名*/
   bp=boy;                            /*bp 指针指向结构体数组首地址*/
   if((fp=fopen(filename,"wb+"))==NULL)
   {  printf("Cannot open file!");
      getch();
      exit(1);
   }
   printf ("input 2 boy information:");
   printf("\nname  num  age  addr:\n");
   for(i=0;i<2;i++,bp++)
       scanf("%s%d%d%s",bp->name,&bp->num,&bp->age,bp->addr);
   bp=boy;
   for(i=0;i<2;i++,bp++)                  /*依次将学生信息存入文件中*/
      fprintf(fp,"%s %d %d %s\n",bp->name,bp->num,bp->age,bp->addr);
   fclose(fp);
   /*以下功能是将信息显示到屏幕上*/
   if((fp=fopen(filename,"r"))==NULL)    /*以只读方式开该文件*/
    {  printf("read cannot open file %s!\n",filename);
       exit(0);
    }
   bp=boy;                                /*bp 指针重新指向结构体数组首地址*/
   for(i=0;i<2;i++,bp++)                  /*将文件中学生信息读取到结构体数组中*/
      fscanf(fp,"%s %d %d %s\n",bp->name,&bp->num, &bp->age,bp->addr);
   printf("name\t  number    age     addr\n");
   bp=boy;
   for(i=0;i<2;i++,bp++)
      printf("%s\t%5d%10d%12s\n",bp->name,bp->num,bp->age,bp->addr);
   fclose(fp);
}
```

调试运行程序

```
Please input the write filename:c:\file4.txt
input 2 boy information:
name  num  age  addr:
zhao  1    18    road12_1
wang  2    17    road14_6
name      number    age      addr
zhao        1        18    road12_1
wang        2        17    road14_6
_
```

小讲堂

fprintf 函数和 fscanf 函数是文件的格式化读/写函数。

7．fprintf函数

fprintf 函数与 printf 函数作用相仿，都是格式化写入函数，只不过写入对象不是终端而是文件。其调用格式为：

```
fprintf(文件指针，格式字符串，输出表列);
```

该函数的功能是按格式字符串的格式，将输出表列的值写入文件指针所指向的文本文件中。注意：输出表列为变量名，不是变量的地址，不能在变量前加“&”符号。例如：

```
fprintf(fp,"%d,%6.2f",i,t);
```

它的作用是将整型变量 i 和实型变量 t 的值按%d 和%6.2f 格式写入 fp 所指向的文件中。如果 i=3，t=4.5，则写入到文件中的是以下字符串（“□”表示空格）：

```
3,□□4.50
```

该函数执行成功，返回值为实际写入的字符个数，否则为负数。

小讲堂

8．fscanf函数

fscanf 函数与 scanf 函数作用相仿，都是格式化读取函数，只不过不是从终端读取而是从文件读取。其调用格式为：

```
fscanf(文件指针，格式字符串，输入表列);
```

该函数的功能是从文件指针所指向的文本文件中读取数据，按格式字符串的格式存入输入表列变量所指向的内存中。注意：输入表列为变量的地址，除字符串输入（用字符数组名接收该字符串）不加“&”符号外，其他变量名前必须加“&”符号。例如：

```
fscanf(fp,"%d,%f",&i,&t);
```

如果文件中有如下字符：

```
3,4.5
```

则文件中的数据 3 送给变量 i，4.5 送给变量 t。函数执行成功，返回值为实际读取的项目的个数，否则为 EOF 或 0。

使用 fscanf 函数需要注意的是，fscanf 从文件中读取数据时，以制表符、空格字符、回车符作为数据项的结束标志。因此，在用 fprintf 函数写入文件时，也要注意在数据项之间留有制表符、空格字符和回车符。

小讲堂

9．文件的其他读写函数

（1）判断文件是否结束函数 feof()，函数调用格式如下：

```
feof(fp);
```

其中，fp 是文件指针。函数功能为判断文件指针 fp 所指文件是否结束，如果遇到文件结束，

函数返回值为 1，否则函数返回值为 0。

（2）文件读/写整数函数，putw 和 getw。对于大多数 C 编译系统来说，都提供这两个函数，用来对磁盘读/写一个字（整数）。getw 函数的调用格式为：

```
整型变量名=getw(文件指针名);
```

putw 函数的调用格式为：

```
putw(整型常量，文件指针名);
```

例如：

```
putw(10,fp);
```

作用是将整数 10 写入 fp 所指的文件，而语句：

```
i=getw(fp);
```

作用是从文件读一个整数到内存，赋给变量 i。

如果所用的 C 编译系统的库函数中不包括 putw 和 getw 函数，则可以自己定义这两个函数。putw 函数定义如下：

```
putw(int  i,FILE  *fp)
{  char  *s;
   s=&i;
   putc(s[0],fp);
   putc(s[1],fp);
   return(i);
}
```

在程序中 s 是指向整型 i 的指针变量，因此 s 指向 i 的第一个字节，s+1 指向 i 的第二个字节。由于*(s+0)就是 s[0]，*(s+1)就是 s[1]，所以 s[0]、s[1]分别对应 i 的第一个字节和第二个字节，顺序输出 s[0]、s[1]就相当于输出了 i 的两个字节中的内容。

同样，可以定义 getw 函数如下：

```
getw(FILE  *fp)
{  char  *s;
   int  i;
   s=&i;
   s[0]=getc(fp);
   s[1]=getc(fp);
   return(i);
}
```

putw 和 getw 并不是 ANSI C 标准定义的函数。但许多 C 编译系统都提供这两个函数，也有的 C 编译系统不用 putw 和 getw 命名，而用其他函数名，使用时请注意。

练一练

【练习 10-4】 编写一个程序：从键盘输入一个字符串，将其中的小写字母全部转换成大写字母，输出到磁盘文件"c:\upper.txt"中保存。输入的字符串以"!"结束。然后再将文件"c:\upper.txt"中的内容读出显示在屏幕上。

解：源程序如下：

```
#include  "stdio.h"
main()
```

```
{  FILE  *fp;
   char  str[100],filename[10];
   int  i=0;
   if((fp=fopen("c:\\upper.txt","w"))==NULL)
   {  printf("Can not open file\n");
      exit(0);
   }
   printf("Enter a string:\n");
   gets(str);
   while(str[i]!='!')
   {  if(str[i]>='a'&&str[i]<='z')
        str[i]=str[i]-32;
      fputc(str[i],fp);
      i++;
   }
   fclose(fp);
   fp=fopen("c:\upper.txt","r");
   fgets(str,strlen(str)+1,fp);
   printf("%s\n",str);
   fclose(fp);
}
```

程序运行结果如下：



本讲小结

本讲主要介绍了文件的各种读写函数，主要有 fputc 函数和 fgetc 函数、fputs 函数和 fgets 函数、fprintf 函数和 fscanf 函数、fwrite 函数和 fread 函数。要注意各种函数的使用格式，如何获得系统的文件名，如何将信息存入文件和从文件中读取信息。

想一想

本讲介绍的函数比较多，在编写程序过程中什么时候使用 fputc 函数和 fgetc 函数、什么时候使用 fputs 函数和 fgets 函数、什么时候使用 fprintf 函数和 fscanf 函数、什么时候使用 fwrite 函数和 fread 函数、并注意各函数在使用时的格式及参数。

第二十七讲　文件的定位和文件的检测

学习目标

- 掌握文件的定位函数;
- 掌握文件的检测函数。

一、文件的定位函数

【思考题 10-6】　如果想将【思考题 10-4】的程序生成的"c:\file3.txt"文件中第奇数个学生的信息输出，应该怎样实现？

程序分析

首先使用 fopen()函数以二进制只读"rb"的方式打开该文件，然后采用重定位函数对文件的指针进行定位，每次指针位置移动两个结构体大小，直到全部输出为止。

编写程序代码

```
#include "stdio.h"
#include "string.h"
struct  student                                    /*定义学生结构体类型*/
{  int  num;                                       /*定义学号*/
   char  name[20];                                 /*定义姓名*/
   int  chinese,math,english;                      /*定义语文，数学，英语成绩*/
};
main()
{  FILE  *fp;
   int  i;
   char  ch,filename[30];
   float  f;
   struct  student  stu[30];                       /*定义结构体类型的变量*/
   clrscr();
   printf("\nPlease input the write filename:");
   scanf("%s",filename);                           /*从键盘读入文件名*/
   if((fp=fopen(filename,"rb"))==NULL)              /*以只读方式打开文件*/
   {  printf("Cannot open %s file\n",filename);
      exit(0);
   }
   printf("\nInformation of the  student score:");
   printf("\n number   name  chinese   math   english\n");
   for(i=0;i<3;i+=2)
   {  fseek(fp,i*sizeof(struct student),0);        /*对文件指针重定位*/
      fread(&stu[i],sizeof(struct student),1,fp); /*读出第 i 个学生信息*/
      printf("\n%5d%9s%7d%9d%7d",stu[i].num,stu[i].name,stu[i].chinese,
             stu[i].math,stu[i].english);
   }
   fclose(fp);
}
```

调试运行程序

输入要打开的文件名，输入"c:\file3.txt"，程序显示运行结果如下：

```
Please input the write filename:c:\file3.txt

Information of the  student score:
 number   name  chinese   math   english

    1     liu    89    66    94
    3    zhao    87    92    90_
```

小讲堂

文件中有一个位置指针，指向当前读写的位置。如果顺序读写一个文件，每次读写一个字符，则读写完一个字符后，位置指针自动移动，指向下一个字符位置。在实际问题中，常

要求读写文件中某些指定的部分。为了避免不必要的读或写的操作，可先移动文件的位置指针到需要读写的位置，再进行读写，这种读写操作方式称为**随机读写**。移动文件位置指针的操作称为**文件的定位**。

实现随机读写的关键是要按指定的条件进行文件的定位操作。文件定位操作是通过库函数的调用来完成的。

1．文件指针重返到文件头部函数rewind

（1）函数调用格式如下：

```
rewind(文件指针);
```

（2）功能。rewind 函数使文件指针指定的文件的位置指针指向文件的开始位置（文件头）。该语句的功能在后面“练一练”中已举了例子进行讲解。



2．返回文件当前指针位置函数ftell

（1）函数调用格式如下：

```
ftell(文件指针);
```

（2）功能。可以用 ftell 函数来返回文件指针的当前位置。

（3）说明。由于在文件的随机读写过程中，位置指针不断移动，往往不容易搞清当前位置，这时就可以使用 ftell 函数得到文件指针的当前位置。ftell 函数的返回值为一个长整型数，表示当前位置相对文件头的字节数，出错时返回-1L。例如：

```
long  i;
if((i=ftell(fp))==-1L)
   printf("A file error has occurred at %ld.\n",i);
```

该程序可通知用户在文件什么位置出现了文件错误。该语句的功能在后面“练一练”中有例子进行讲解。



3．移动文件指针到指定位置fseek函数

（1）函数调用格式如下：

```
fseek(文件指针,位移量,起始位置);
```

（2）功能。将文件指针指到以“起始位置”定位点为初始位置，移动“位移量”字节后的位置上。如果文件定位成功，则 fseek 返回 0，否则返回一个非 0 值。

说明：

- 起始位置分别有 0（文件开始）、1（文件当前位置）和 2（文件末尾）三种情况。
- 位移量可正可负。位移量为正数时，位置指针向后移动；位移量为负数时，位置指针向前移动。

“起始位置”是初始位置的“定位代码”，表示从文件的哪一点开始测量“位移量”，

按表 10-2 所示的规定方式取值，既可以是标准 C 规定的常量名，也可以取对应的数字。

fseek 函数常用于二进制文件的随机读写。用于文本文件时，因字符转换问题，常出现定错位问题。

表 10-2　指针初始位置表示法

符号名	数　字	含　义
SEEK_SET	0	文件开头
SEEK_CUR	1	文件指针当前位置
SEEK_END	2	文件末尾

例如：

```
fseek(fp,58L,0);    /*文件指针从文件开始处向后移动 58 字节*/
fseek(fp,30L,1);    /*文件指针从当前位置向后移动 30 字节*/
fseek(fp,-15L,2);   /*文件指针从文件末尾处向前移动 15 字节*/
```

注意　位移量一般被要求是长整（long）型数据，这样当文件的长度大于 64KB 时不致出问题。

在本讲【思考题 10-6】例程中的语句：

```
fseek(fp,i*sizeof(struct student),0);        /*对文件指针重定位*/
```

就是将文件指针定位到第奇数位的首地址处（因为 i 每次自加 2）。

二、文件的检测函数

【思考题 10-7】　当要打开的文件不存在时，显示错误提示信息，若存在时，输出文件中的所有信息，程序应该如果编写？

程序分析

首先使用 fopen()函数以只读"rb"的方式打开该文件，若文件不存在，则给出提示信息（在此处使用 ferror 函数和 clearerr 函数）。若存在，则将该文件的信息全部输出，直到文件结束为止。

编写程序代码

```
#include  "stdio.h"
main()
{  char  c,filename[30];
   FILE *fp;
   clrscr();                          /*清屏*/
   printf("\nPlease input the open filename:");
   scanf("%s",filename);              /*从键盘读入文件名*/
   fp=fopen(filename,"r");
   if(ferror(fp))                     /*若文件指针出错，即没打开该文件*/
   {  printf("Error reading from %s file\n",filename);
      clearerr(fp);                   /*将文件出错指针置为 0*/
      exit(0);
   }
   else
   {  while ((c=fgetc(fp))!=EOF) /*当文件指针没到文件尾部时*/
```

```
            putchar(c);
        fclose(fp);                    /*关闭该文件*/
    }
}
```



调试运行程序

当输入的文件名存在时，程序会输出该文件中的所有字符，运行结果如下：

```
Please input the open filename:c:\file1.txt
ABC_
```

当输入的文件名不存在时，程序会给出错误提示信息，运行结果如下：

```
Please input the open filename:c:\aaa.txt
Error reading from c:\aaa.txt file
Null pointer assignment
_
```



C 标准中有一些检测输入/输出函数调用中的错误的函数，主要有文件结束检测函数 feof()、文件出错检测函数 ferror()及文件出错标志和文件结束标志置 0 函数 clearerr()三个。

4．文件结束检测函数feof()

（1）函数调用格式如下：

```
int    feof(文件指针);
```

（2）功能。feof()函数用来判断“文件指针”指向的文件是否处于文件结束位置，如文件结束，则返回值为 1，否则为 0。



5．文件出错检测函数ferror()

大多数输入/输出函数不具有明确的出错信息返回，在调用各种输入/输出函数（如 fputc、fgetc、fread、fwrite 等）时，如果出现了错误，除了函数返回值有所反映外，还可以用 ferror 函数检测。

（1）函数调用格式如下：

```
int    ferror(文件指针);
```

（2）功能。ferror()函数用来检查文件 fp 在用各种输入/输出函数进行读/写时是否出错，若出错，返回值为 1，否则返回 0。

说明：

- ferror 函数的返回值为 0，则表示未出错；如果返回非零值，表示出错。
- 在调用 fopen 函数时，会自动使相应文件的 ferror 函数的初值为零。
- 应注意，每调用一次输入/输出函数后，都有一个 ferror 函数值与之对应，如果想检查调用输入/输出函数是否出错，应该在调用该函数后立即测试 ferror 函数的值，否则该值会丢失。

- ferror 函数反映的是最后一个函数调用的出错状态。
- 在执行 fopen 函数时，ferror 函数的初值自动置为 0。



6．文件出错标志和文件结束标志置 0 函数clearerr()

（1）函数调用格式如下：

```
int   clearer(文件指针);
```

（2）功能。clearer()函数用来使文件的错误标志和文件结束标志置为 0。假设在调用一个输入/输出函数时出现错误，ferror 函数值为一个非零值。在调用 clearerr(fp)后，ferror(fp)的值变成 0。



7．常用文件操作函数表

C 语言对文件的操作是通过调用库函数来完成的。下面将本章介绍过的输入/输出函数做概括性小结，具体如表 10-3 所示。

表 10-3　常用文件操作函数

文件读/写	fgetc(),getc()	从指定文件读取一个字符
	fputc(),putc()	把字符写入指定文件
	fgets()	从指定文件读取一个字符串
	fputs()	把字符串写入指定文件
	getw()	从指定文件读取一个字（int 型）
	putw()	把字（int 型）写入指定文件
	fread()	从指定文件读取数据项
	fwrite()	把数据项写入指定文件
	fscanf()	从指定文件中按指定格式输入数据
	fprintf()	按指定格式将数据输出到指定文件中
文件定位	rewind()	文件指针重返到文件头部
	ftell()	返回文件当前指针位置
	fseek()	移动文件指针到指定位置
文件检测	feof()	若到文件末尾，函数值为真（非 0）
	ferror()	若对文件操作出错，函数值为真（非 0）
	clearerr()	使 ferror 和 feof 函数值置 0

练一练

【练习 10-5】　设计一个程序实现人员登录，即每当从键盘接收一个姓名时，便在文件"c:\member.dat"中进行查

找。若此姓名已存在，则显示相应信息，若文件中没有该姓名，则将其存入文件（若文件"c:\member.dat"不存在，应在磁盘上建立一个新文件）。当输入姓名按<回车>键或处理过程中出现错误时程序结束。

解：源程序如下：

```
#include  "stdio.h"
main()
{  FILE  *fp;
   int flag;
   char  name[20],data[20];
   if((fp=fopen("c:\\member.dat","a+"))==NULL)   /*以追加方式打开文件*/
   {  printf("Open file error\n");
      exit(0);
   }
   do
   {  printf("Enter name(end with input empty):\n");
      gets(name);                        /*读入字符串*/
      if(strlen(name)==0)                /*当输入名字为空时跳出循环*/
         break;
      strcat(name,"\n");                 /*为字符串加一个回车符*/
      rewind(fp);                        /*文件指针重新回到文件头*/
      flag=1;
      while(flag&&(fgets(data,30,fp)!=NULL))
      /*当输入名字正确且从文件中读数据正确时*/
        if(strcmp(data,name)==0)         /*若输入名字与已存名字相同*/
           flag=0;
     if(flag)
       fputs(name,fp);
     else
       printf("\nThis name has been existed!\n");
       }while(ferror(fp)==0);              /*当文件指针不出错时*/
     fclose(fp);
}
```

程序运行结果如下：

```
Enter name(end with input empty):
liu

This name has been existed!
Enter name(end with input empty):
wang

This name has been existed!
Enter name(end with input empty):
zhao
Enter name(end with input empty):
li

This name has been existed!
Enter name(end with input empty):
liu

This name has been existed!
Enter name(end with input empty):
```

本讲小结

本讲主要介绍了：（1）文件的定位函数，包括三个：rewind 函数、ftell 函数和 fseek 函数，（2）文件

的检测函数，包括三个：文件结束检测 feof 函数、文件出错检测 ferror 函数及文件出错标志和文件结束标志置 0 的 clearerr 函数。要学会这些函数的格式及功能，以及要注意在程序中使用时的参数类型。

想一想

在编写程序过程中，什么时候要用到文件的定位函数，什么时候要用到文件的检测函数。

本章小结

本章着重介绍了文件、文件系统、文件指针的概念，文件的打开和关闭，文件的基本操作、文件的定位，文件的检测等内容。特别是文件的字符级、字符串级、数据块级的输入/输出函数，在文件操作中非常重要。

磁盘文件可以分为文本文件和二进制文件两种存储方式，每个磁盘文件都有一个文件结束符标记文件结束位置。

无论是文本文件还是二进制文件，在访问它之前都要先打开文件，然后才能访问该文件，对文件操作结束后，再关闭该文件。

对文件的访问操作包括输入和输出两种操作，输入操作是指从外部文件向内存变量输入数据，输出操作是指把内存变量或表达式的值写入到外部文件中。

通过学习，读者应全面掌握以上内容，结合前几章所学的各种算法，熟练地编写各种文件类型的程序。

课后习题十

一、选择题

1．系统的标准输入文件是指_____。

A）键盘　　B）显示器　　C）软盘　　D）硬盘

2．若执行 fopen 函数时发生错误，则函数的返回值是_____。

A）地址值　　B）0　　C）1　　D）EOF

3．若要用 fopen 函数打开一个新的二进制文件，该文件要既能读能也写，则文件方式字符串应是_____。

A）"ab+"　　B）"wb+"　　C）"rb+"　　D）"ab"

4．当顺利执行了文件关闭操作时，fclose 函数返回值是_____。

A）-1　　B）TURE　　C）0　　D）1

5．已知函数的调用形式：fread(buffer,size,count,fp);，其中 buffer 代表的是_____。

A）一个整型变量，代表要读入的数据项总数

B）一个文件指针，指向要读的文件

C）一个指针，指向要读入数据的存放地址

D）一个存储区，存放要读的数据项

6．rewind 函数的作用是_____。

A）重新打开文件

B）使文件位置指针重新回到文件末尾

C）使文件位置指针重新回到文件的开始

D）返回文件长度值

7．标准函数 fgets(s,n,f)的功能是_____。

A）从文件 f 中读取长度为 n 的字符串存入指针 s 所指的内存

B）从文件 f 中读取长度不超过 n−1 的字符串存入指针 s 所指的内存

C）从文件 f 中读取 n 个字符串存入指针 s 所指的内存

D）从文件 f 中读取长度为 n−1 的字符串存入指针 s 所指的内存

8．若 fp 是指向某文件的指针，且已读到文件末尾，则库函数 feof(fp)的返回值是____ 。

A）EOF　　B）−1　　C）非零值　　D）NULL

9．若要打开 A 盘上 user 子目录下名为 abc.txt 的文本文件进行读、写操作，下面符合此要求的函数调用是_____。

A）fopen("A:\user\abc.txt","r")　　B）fopen("A:\\user\\abc.txt","r+")

C）fopen("A:\user\abc.txt","rb")　　D）fopen("A:\\user\\abc.txt","w")

10．以数据块为单位对数据进行整体读/写时如果 ptr 是指向内存中数据块的首地址，fp 是文件指针，那么数据块中每个数据项的大小为_____。

A）sizeof(*ptr)　　B）*ptr　　C）sizeof(ptr)　　D）sizeof(*fp)

11．利用 fseek 函数可实现的操作是_____。

A）改变文件的位置指针　　B）文件的顺序读/写

C）文件的随机读/写　　D）以上答案均正确

12．函数调用语句 fseek(fp,20L,2);的含义是_____。

A）将文件位置指针移动距离文件头 20 字节处

B）将文件位置指针从当前位置向后移动 20 字节

C）将文件位置指针从文件末尾向后退 20 字节

D）将文件位置指针移到离当前位置 20 字节处

13．函数 ftell 的作用是_____。

A）得到流式文件中的当前位置　　B）移动流式文件的位置指针

C）初始化流式文件的位置指针　　C）以上答案均正确

14．在执行 fopen 函数时，ferror 函数的初值是_____。

A）TURE　　B）−1　　C）1　　D）0

15．设有以下结构体类型：

```
struct  st
{  char  name[8];
   int  num;
   float  s[4];
}student[50];
```

并且结构体数组 student 中的元素都已有值，若要将这些元素写到硬盘文件 fp 中，以下不正确的形式是_____。

A）fwrite(student,sizeof(struct st),50,fp);

B）fwrite(student,50*sizeof(struct st),1,fp);

C）fwrite(student,25*sizeof(struct st),25,fp);

D）for(i=0;i<50;i++)

fwrite(student+i,sizeof(struct st),1,fp);

二、填空题

1．在 C 程序中，文件可以用_____方式存取，也可以用_____方式存取。

2．在 C 程序中，数据可以用_____和_____两种代码形式存放。

3．在 C 语言中，文件的存取是以_____为单位的，这种文件被称为_____文件。

4．函数调用语句 fgets(buf,n,fp);从 fp 指向的文件中读入_____个字符放到 buf 字符数据中，函数值为_____。

5．下面程序用变量 count 统计文件中字符的个数，请在空白处填入适当的内容。

```
#include "stdio.h"
main()
{ FILE *fp;
  long count=0;
  if((fp=fopen("letter.dat",  【1】  ))==NULL)
  { printf("can not open file!\n"); exit(0);}
  while(!feof((fp))
  {   【2】  ;  【3】 ;}
  printf("count=%ld\n",count);
  fclose(fp);
}
```

【1】_____【2】_____【3】_____

三、编程题

1．设文件 number.dat 中存放了一组整数。请编程统计并输出文件中的正整数、零和负整数的个数。

2．设文件 student.dat 中存放着一年级学生的基本情况，这些情况由以下结构体来描述：

```
struct student
{ long int num;        /*学号*/
  char name[10];       /*姓名*/
  int age;             /*年龄*/
  char sex;            /*性别*/
  char speiality[20];  /*专业*/
  char addr[40];       /*住址*/
};
```

请编程序，输出学号在 970101～970135 之间的学生学号、姓名、年龄和性别。

第十一章　库函数及应用

本章主要介绍了常用的字符屏幕处理函数和图形处理函数。正因为有了这些函数，我们才能编写出图形化的用户界面，既美观又方便用户。

Turbo C 提供了字符屏幕和图形函数。字符屏幕的核心是窗口（Window），它是屏幕的活动部分，字符输出或显示在活动窗口中进行。默认时，窗口就是整个屏幕。窗口可以根据需要指定其大小。同样，对图形函数的操作，也提供了视口（Viewport），也就是说图形函数的操作都是在视口上进行的。图形视口与字符窗口具有相同的特性，用户可以在屏幕上定义大小不同的视口，若不定义视口大小，它就是整个屏幕。

窗口是在字符屏幕下的概念，只有字符才能在窗口中显示出来，这时用户可以访问的最小单位为一个字符。视口是在图形屏幕状态下的概念，字符与图形都可以在视口上显示，用户可访问的最小单位是一个像素（像素这一术语最初用来指显示器上最小的、单独的发光点单元。然而现在，其含义拓宽为指图形显示器上的最小可访问点）。

字符和图形状态下，屏幕上的位置都是由它们的行和列所决定的。有一点须指出：字符状态左上角坐标为(1,1)，但图形左上角坐标为(0,0)。

了解字符屏幕和图形函数与窗口和视口的关系是很重要的。例如，字符屏幕光标位置函数 gotoxy()将光标移到窗口的(x,y)位置上，这未必是相对于整个屏幕。

函数原型和字符屏幕处理函数的头部信息在 conio.h 中，所有图形系统的有关信息和原型在 graphics.h 中。

第二十八讲　字符屏幕处理函数



学习目标

- 掌握字符窗口颜色的设置;
- 掌握窗口字符的输入/输出函数;
- 掌握窗口字符的其他处理函数。

常用的字符屏幕处理函数主要包括字符窗口大小的设定、窗口颜色的设置、窗口字符输入/输出和窗口字符的清除等函数。字符屏幕的所有函数都包含在 conio.h 中，请注意在程序中包含该头文件。下面介绍几类常用的字符屏幕函数的功能、操作方法。

一、字符窗口的定义

【思考题 11-1】　分析以下程序的功能，注意其中的窗口定义和颜色设置函数。



程序分析

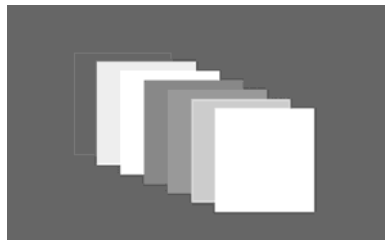
该程序使用了关于窗口大小的定义、颜色的设置等函数，在一个屏幕上不同位置定义了 7 个窗口，其背景色分别使用了 7 种不同的颜色。

编写程序代码

```
#include  <stdio.h>
#include  <conio.h>
main()
{  int  i;
   textbackground(0);                          /*设置屏幕背景色*/
   clrscr();                                   /*清除字符屏幕*/
   for(i=1;i<8;i++)
   {  window(10+i*5,5+i,30+i*5,15+i);          /*定义字符窗口*/
   textbackground(i);                          /*定义窗口背景色*/
   clrscr();                                   /*清除窗口*/
}
}
```

调试运行程序

程序运行结果如下：



小讲堂

1．字符窗口介绍

Turbo C 2.0 默认定义的字符窗口为整个屏幕，共有 80 列（或 40 列）25 行的字符单元，每个单元包括一个字符和一个属性，字符即 ASCII 码字符，属性规定该字符的颜色和强度。

Turbo C 2.0 可以定义屏幕上的一个矩形域作为窗口，使用 window()函数定义。窗口定义之后，用有关窗口的输入/输出函数就可以只在此窗口内进行操作而不超出窗口的边界。

window()函数的调用格式为：

```
void    window(int   left, int   top, int   right, int   bottom);
```

其中，形式参数(int left, int top)是窗口左上角的坐标，(int right, int bottom)是窗口的右下角坐标，(left, top)和(right, bottom) 是相对于整个屏幕而言的。

Turbo C 2.0 规定整个屏幕的左上角坐标为(1,1)，右下角坐标为（80,25），并规定沿水平方向为 X 轴，方向朝右；沿垂直方向为 Y 轴，方向朝下。若 window()函数中的坐标超过了屏幕坐标的界限，则窗口的定义就失去了意义，也就是说定义将不起作用，但程序编译链接时并不出错。

另外，一个屏幕可以定义多个窗口，但现行窗口只能有一个（因为 DOS 为单任务操作系统），当需要用另一窗口时，可将定义该窗口的 window()函数再调用一次，此时该窗口便成为现行窗口了。

如要定义一个窗口左上角在屏幕(20,5)处，大小为 30 列 15 行的窗口可写成：

```
window(20,5,50,20);
```



2．字符窗口颜色的设置

字符窗口颜色的设置包括背景颜色的设置和字符颜色的设置，使用的函数及其调用格式为：

设置背景颜色：

```
void  textbackground(int  color);
```

设置字符颜色：

```
void  textcolor(int  color);
```

有关颜色的定义如表 11-1 所示。

表 11-1　颜色值

符 号 常 数	数　值	含　义	字符或背景
BLACK	0	黑	两者均可
BLUE	1	蓝	两者均可
GREEN	2	绿	两者均可
CYAN	3	青	两者均可
RED	4	红	两者均可
MAGENTA	5	洋红	两者均可
BROWN	6	棕	两者均可
LIGHTGRAY	7	淡灰	两者均可
DARKGRAY	8	深灰	只用于字符
LIGHTBLUE	9	淡蓝	只用于字符
LIGHTGREEN	10	淡绿	只用于字符
LIGHTCYAN	11	淡青	只用于字符
LIGHTRED	12	淡红	只用于字符
LIGHTMAGENTA	13	淡洋红	只用于字符
YELLOW	14	黄	只用于字符
WHITE	15	白	只用于字符
BLINK	128	闪烁	只用于字符

表 11-1 中的符号常数与相应的数值等价，二者可以互换。例如设定蓝色背景可以使用 textbackground(1)，也可以使用 textbackground(BLUE)，两者没有任何区别，只不过后者比较容易记忆，一看就知道是蓝色。

另外，Turbo C 还提供了一个函数，可以同时设置字符的字符和背景颜色，这个函数的调用格式为：

```
void  textattr(int  attr);
```

其中：attr 的值表示颜色形式编码的信息，每一位代表的含义如下：

```
数位：    7   6   5   4   3   2   1   0
码：      B   b   b   b   c   c   c   c
          ↑   └──┘   └──────┘
```

闪烁　背景颜色　　字符颜色

字节低 4 位 cccc 设置字符颜色（0～15），4～6 位中的 3 位 bbb 设置背景颜色（0～7），第 7 位 B 设置字符是否闪烁。例如：要设置一个蓝底黄字，定义方法如下：

```
textattr(YELLOW+(BLUE<<4));
```

若再要求字符闪烁，则定义变为：

```
textattr(128+YELLOW+(BLUE<<4);
```



注意

（1）对于背景只有 0 到 7 共 8 种颜色，若取大于 7 小于 15 的数，则代表的颜色与减 7 后的值对应的颜色相同。

（2）用 textbackground()和 textcolor() 函数设置了窗口的背景与字符颜色后，在没有用 clrscr()函数清除窗口之前，颜色不会改变，直到使用了函数 clrscr()，整个窗口和随后输出到窗口中的字符字符才会变成新颜色。

（3）用 textattr()函数时背景颜色应左移 4 位，才能使 3 位背景颜色移到正确位置。

二、字符窗口的输入/输出函数

【思考题 11-2】以【思考题 11-1】为基础，加入窗口内字符输出函数。



程序分析

该程序使用了关于窗口大小的定义、颜色的设置等函数，在一个屏幕上的不同位置定义了 7 个窗口，其背景色分别使用了 7 种不同的颜色，并在最后一个窗口内的不同位置显示了 8 个字符串。



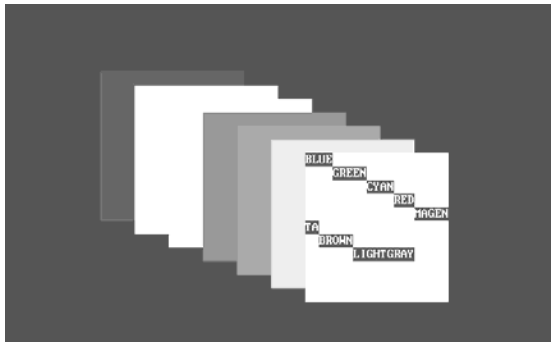
编写程序代码

```
#include  <stdio.h>
#include  <conio.h>
main()
{  int  i;
   char*c[]={"BLACK","BLUE","GREEN","CYAN","RED","MAGENTA","BROWN",
             "LIGHTGRAY"};
   textbackground(0);                  /* 设置屏幕背景色 */
   clrscr();                           /* 清除字符屏幕 */
   for(i=1; i<8; i++)
   {  window(10+i*5,5+i,30+i*5,15+i); /* 定义字符窗口 */
      textbackground(i);               /* 定义窗口背景色 */
      clrscr();                        /* 清除窗口 */
    }
    textbackground(0);                 /* 设置输出字符的背景颜色为黑色 */
    for(i=1;i<8;i++)                   /* 在最前面窗口中输出各字符串 */
    {  cputs(c[i]);
       putch('\n');
    }
}
```



调试运行程序

程序运行结果如下：



小讲堂

3．窗口内字符的输出函数

窗口内字符的输出函数有 3 种，函数原型分别如下：

```
int   cprintf("格式化字符串",输出变量列表);
int   cputs(char   *string);
int   putch(int   ch);
```

说明：

（1）cprintf()函数输出一个格式化的字符串或数值到窗口中。它与 printf()函数的用法完全一样，区别在于 cprintf() 函数的输出受窗口限制，而 printf()函数的输出为整个屏幕。

（2）cputs()函数输出一个字符串到屏幕上，它与 puts()函数用法完全一样，只是受窗口大小的限制。

（3）putch()函数输出一个字符到窗口内。

（4）使用上述输出函数，当输出超出窗口的右边界时会自动移到下一行的开始处继续输出。当窗口内已填满内容仍没有结束输出时，窗口屏幕将会自动逐行上卷，直到输出结束为止。

小讲堂

4．窗口内字符的输入函数

窗口内字符的输入函数有 4 种，函数原型分别如下：

```
int   getch(void);
int   getche(void);
char   *cgets(char   *str);
int   cscanf("格式控制字符串",输入地址列表);
```

说明：

（1）getch()函数由键盘读取一个字符，并将该字符显示在窗口内，在屏幕上显示的时候，如果字符超过了屏幕右边界，会被自动转移到下一行的开始位置。

（2）窗口内字符的输入函数 int getche(void);从键盘上获得一个字符，在屏幕上显示的时候，如果字符超过了窗口右边界，则会被自动转移到下一行的开始位置。

（3） cgets()函数由键盘读取字符串，cscanf()由键盘格式化输入，它们和 gets()、scanf()

函数相对应，只不过是针对窗口范围进行输入操作。

三、字符窗口的屏幕操作函数

【思考题 11-3】 设计一个程序，使用屏幕操作的各函数，在屏幕上显示背景色为蓝色，窗口背景色为红色，窗口内字符颜色为黄色的多行字符串。

程序分析

因为要设置字符颜色、整个屏幕背景颜色，所以要用到函数 textcolor()和 textbackground()函数。又因为要使用窗口，所以要用到窗口函数 window()。该程序还用到了 getxtext()、movetext()、puttext()、getch()、gotoxy()等函数。请注意这些函数的使用方式。

编写程序代码

```
#include  "conio.h"
main()
{  int  i;
   char  *f[]={"Load F3","Pick Alt-F3","New ","Save F2","Write to ","Direc
               tory","Change dir","Os shell ","Quit Alt-X"};
   char  buf[11*16*2];
   clrscr();
   textcolor(YELLOW);                /*设置字符颜色为黄色*/
   textbackground(BLUE);             /*设置字符背景颜色为蓝色*/
   clrscr();
   gettext(10, 2, 24, 11, buf);
   window(10, 2, 24, 11);
   textbackground(RED);              /*设置窗口内的字符颜色为红色*/
   textcolor(YELLOW);
   clrscr();
   for(i=0; i<9; i++)
   {   gotoxy(1,i+1);
       cprintf("%s", f[i]);
   }
   getch();
   movetext(10, 2, 24, 11, 40, 10);
   puttext(10, 2, 24, 11, buf);
   getch();
}
```

调试运行程序

程序运行结果如下：



小讲堂

5．有关屏幕操作的函数

（1）窗口字符清除函数 clrscr()。

函数原型：

```
void    clrscr(void) ;
```

函数功能：清除当前窗口中的字符内容，并把光标定位在窗口的左上角(1,1)处。

（2）窗口字符行清除函数 clreol()。

函数原型：

```
void clreol(void);
```

函数功能：清除当前窗口中从光标位置到行尾的所有字符，光标位置不变。

（3）窗口光标定位函数 gotoxy()。

函数原型：

```
void gotoxy(x,y);
```

函数功能：该函数很有用，它用来定位光标在当前窗口中的位置。这里(x,y)是指光标要定位处的坐标（相对于窗口而言），当(x,y)超出了窗口的大小时，该函数就不起作用了。

（4）窗口字符读取函数 gettext()。

函数原型：

```
int gettext(int xl,int yl,int x2,int y2,void *buffer);
```

函数功能：将屏幕上指定的矩形区域内字符内容存入 buffer 指针指向的一个内存空间。内存的大小用下面的公式计算：

```
所用字节大小=行数×列数×2
```

其中：行数=y2−y1+1，列数=x2−x1+1。

（5）窗口字符存取函数 gettext()和 puttext()。

函数原型：

```
int gettext(int xl,int yl,int x2,int y2,void *buffer);
```

函数功能：此函数是将用 gettext()函数存入内存 buffer 中的文字内容拷贝到屏幕上指定的位置。

（6）窗口字符拷贝函数 movetext()。

函数原型：

```
int movetext(int x1,int y1, int x2, int y2, int x3, int y3);
```

函数功能：movetext()函数将屏幕上左上角为(x1, y1)，右下角为(x2, y2)的一矩形窗口内的字符内容拷贝到左上角为(x3, y3)的新的位置。该函数的坐标也是相对于整个屏幕而言的。

（7）返回窗口光标位置函数。

函数原型：

```
int wherex(void);
```

和

```
int wherey(void);
```

函数功能：这两个函数用于返回当前窗口中光标的(x,y)坐标。

（8）高/低亮度显示字符函数。

函数原型：

```
void highvideo(void);
```

函数功能：设置显示器高亮度显示字符。

函数原型：

```
void lowvideo(void);
```

函数功能：设置显示器低亮度显示字符。

（9）返回原来显示方式函数 normvideo()。

函数原型：

```
void normvideo(void);
```

函数功能：使显示器返回到程序运行前的显示方式。

对于以上几个函数的两点说明如下：

① gettext()函数和 puttext() 函数中的坐标是对整个屏幕而言的，是屏幕的绝对坐标，而不是相对窗口的坐标；

② movetext()函数是拷贝而不是移动窗口区域内容，即使用该函数后，原位置区域的字符内容仍然存在。

练一练

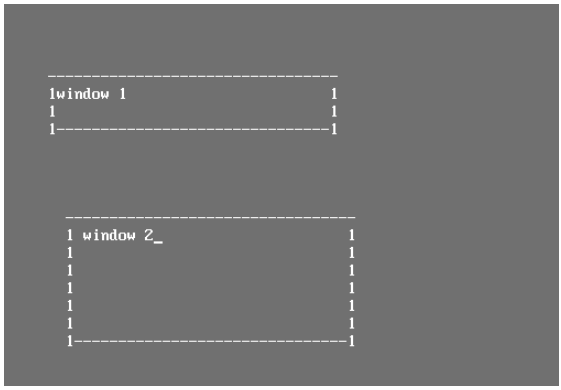
【练习 11-1】 设计一个程序，在屏幕上建立两个窗口，然后在窗口里显示字符。提示，使用窗口光标定位函数 gotoxy()，字符的位置是调用该函数确定的。

解：源程序如下：

```
#include "conio.h"
void border(int startx,int starty,int endx,int endy)
{  register int i;
   gotoxy(1,1);
   for(i=0;i<=endx-startx;i++)
      putch('-');
   gotoxy(1,endy-starty);
   for(i=0;i<=endx-startx;i++)
      putch('-');
   for(i=2;i<=endy-starty;i++)
   {  gotoxy(1,i);
      putch('1');
      gotoxy(endx-startx+1,i);
      putch('1');
   }
}
main()
{  void border(int,int,int,int);
   clrscr();
   window(6,8,38,12);
   border(6,8,38,12);
   gotoxy(2,2);
   printf("window 1");
   window(8,16,40,24);
```

```
        border(8,16,40,24);
        gotoxy(3,2);
        printf("window 2");
        getch();
    }
```

程序运行结果如下：



本讲小结

本讲主要介绍了常用的字符屏幕处理函数，主要包括字符窗口大小的设定，窗口颜色的设置，窗口字符输入/输出和窗口字符的清除等函数。要注意这些函数的调用格式及参数的类型。

想一想

这些字符屏幕处理函数一般用在什么时候？如果想显示图形信息，应该如何处理？

第二十九讲　图形处理函数（一）

学习目标

- 掌握图形模式的初始化;
- 掌握屏幕颜色的设置和清屏函数;
- 掌握基本画图函数。

Turbo C 提供了非常丰富的图形处理函数，所有图形处理函数的原型均在 graphics. h 中，本讲主要介绍图形模式的初始化、独立图形程序的建立、基本图形功能、图形窗口以及图形模式下的字符输出等函数。

另外，使用图形处理函数时要确保有显示器图形驱动程序。*. BGI，同时将集成开发环境 options/Linker 中的 Graphics lib 选为 on，只有这样才能保证正确使用图形处理函数。

一、图形模式的初始化

【思考题 11-4】　使用图形初始化函数设置 VGA 高分辨率图形模式，并画一个长方体。

程序分析

不同的显示器适配器有不同的图形分辨率，即使是同一显示器适配器，在不同模式下也有不同分辨率。因此，在屏幕作图之前，必须根据显示器适配器种类将显示器设置成为某种

图形模式。

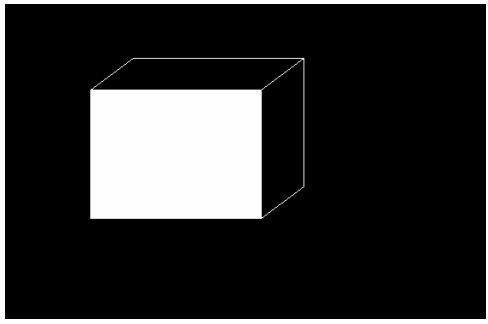
设置屏幕为图形模式，可用图形初始化函数 initgraph()，画长方体函数用 bar3d()即可。注意这些图形函数的使用方法。

编写程序代码

```
#include  <graphics.h>
int  main()
{   int gdriver, gmode;
    gdriver=VGA;
    gmode=VGAHI;
    initgraph(&gdriver, &gmode, "c:\\ tc");
    bar3d(100, 100, 300, 250, 50, 1);      /*画一长方体*/
    getch();
    closegraph();
    return 0;
}
```

调试运行程序

程序运行结果如下：



小讲堂

1．图形模式的初始化

不同的显示器适配器有不同的图形分辨率在屏幕作图之前，必须根据显示器适配器种类将显示器设置成为某种图形模式，在未设置图形模式之前，微机系统默认屏幕为字符模式(80 列,25 行字符模式)，此时所有图形函数均不能工作。设置屏幕为图形模式，可用下列图形初始化函数 initgraph()，其函数调用格式为：

```
void far initgraph(int far *gdriver, int far *gmode, char *path);
```

其中 gdriver 和 gmode 分别表示图形驱动器和模式，path 是指图形驱动程序所在的目录路径。有关图形驱动器、图形模式的符号常数及对应的分辨率如表 11-2 所示。

图形驱动程序由 Turbo C 出版商提供，文件扩展名为.BGI。根据不同的图形适配器有不同的图形驱动程序。例如对于 EGA、VGA 图形适配器就调用驱动程序 EGAVGA.BGI。

表 11-2　图形驱动器、模式的符号常数及数值

图形驱动器（gdriver）		图形模式（gmode）		色　　调	分　辨　率
符号常数	数值	符号常数	数值		
CGA	1	CGAC0	0	C0	320×200
		CGAC1	1	C1	320×200
		CGAC2	2	C2	320×200
		CGAC3	3	C3	320×200
		CGAHI	4	2 色	640×200
MCGA	2	MCGAC0	0	C0	320×200
		MCGAC1	1	C1	320×200
		MCGAC2	2	C2	320×200
		MCGAC3	3	C3	320×200
		MCGAMED	4	2 色	640×200
		MCGAHI	5	2 色	640×480
EGA	3	EGALO	0	16 色	640×200
		EGAHI	1	16 色	640×350
EGA64	4	EGA64LO	0	16 色	640×200
		EGA64HI	1	4 色	640×350
		EGA64HI	1	4 色	640×350
EGAMON	5	EGAMONHI	0	2 色	640×350
IBM8514	6	IBM8514LO	0	256 色	640×480
		IBM8514HI	1	256 色	1024×768
HERC	7	HERCMONOHI	0	2 色	720×348
ATT400	8	ATT400C0	0	C0	320×200
		ATT400C1	1	C1	320×200
		ATT400C2	2	C2	320×200
		ATT400C3	3	C3	320×200
		ATT400MED	4	2 色	320×200
		ATT400HI	5	2 色	320×200
VGA	9	VGALO	0	16 色	640×200
		VGAMED	1	16 色	640×350
		VGAHI	2	16 色	640×480
PC3270	10	PC3270HI	0	2 色	720×350
DETECT	0	用于硬件测试			

有时编程者并不知道所用的图形显示器适配器种类，或者需要将编写的程序用于不同图形驱动器， Turbo C 提供了一个自动检测显示器硬件的函数 detectgraph()， 其调用格式为：

```
void  far  detectgraph(int *gdriver, *gmode);
```

其中 gdriver 和 gmode 的意义与图形初始化函数 initgraph()的参数意义相同。

例如，可以自动进行硬件测试后进行图形初始化，则【思考题 11-4】的程序可改为：

```
#include <graphics.h>
main(){  int gdriver, gmode;
    detectgraph(&gdriver, &gmode);          /*自动测试硬件*/
    printf("the graphics driver is %d, mode is %d\n", gdriver, gmode);
/*输出测试结果*/
    getch();
    initgraph(&gdriver, &gmode,"c:\\tc");  /* 根据测试结果初始化图形*/
    bar3d(100, 100, 300, 250, 50, 1);       /*画一长方体*/
    getch();
    closegraph();
    return 0;
}
```

上面例程中先对图形显示器自动检测，然后再用图形初始化函数进行初始化设置，但 Turbo C 提供了一种更简单的方法，即用 gdriver=DETECT 语句后再跟 initgraph()函数就行了。

另外，Turbo C 提供了退出图形状态的函数 closegraph()，其调用格式为：

```
void far closegraph(void);
```

调用该函数后可退出图形状态而进入字符方式（Turbo C 默认方式）,并释放用于保存图形驱动程序和字体的系统内存。

采用这种方法后, 上面例程又可改为：

```
#include  <conio.h>
#include  <graphics.h>
main()
{  int  gdriver=DETECT, gmode;
   initgraph(&gdriver, &gmode,"c:\\tc");
   bar3d(100, 100, 300, 250, 50, 1);       /*画一长方体*/
   getch();
   closegraph();
   return 0;
}
```

这两个改过的程序的运行结果同【思考题 11-4】。

二、屏幕颜色的设置和清屏函数

【思考题 11-5】 分析以下程序的运行结果，学习怎样使用屏幕颜色函数和图形的清屏函数。



程序分析

首先初始化图形界面，设置图形背景颜色为黑色，然后设置不同的颜色，画半径不同的同心圆，每个圆延迟 20000 毫秒。再重新设置背景色，每次变换背景色，并在该背景色上画半径逐渐变大的圆，最后结束程序。



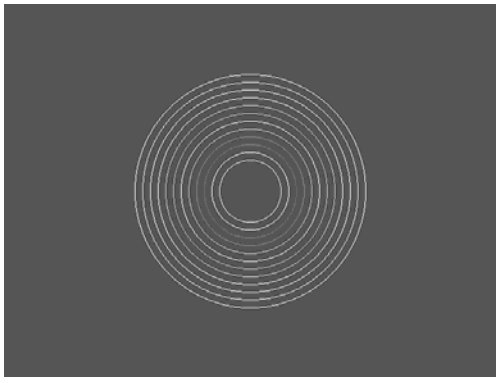
编写程序代码

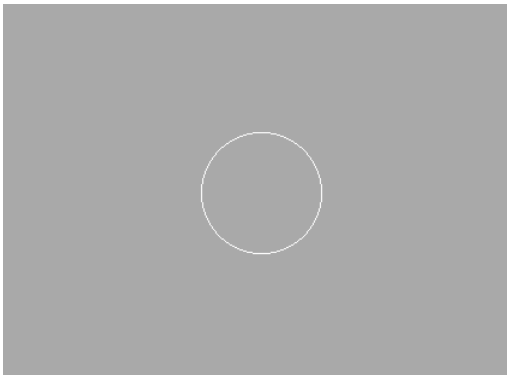
```
#include<stdio.h>
#include<graphics.h>
main()
```

```
{   int  gdriver, gmode, i;
    gdriver=DETECT;
    initgraph(&gdriver,&gmode, "c:\\tc");   /*图形初始化*/
    setbkcolor(0);                          /*设置图形背景*/
    cleardevice();
    for(i=0; i<=15; i++)
    {   setcolor(i);                        /*设置不同作图色*/
        circle(320, 240, 20+i*10);          /*画半径不同的圆*/
        delay(20000);                       /*延迟 20000ms*/
    }
    for(i=0; i<=15; i++)
    {   setbkcolor(i);                      /*设置不同背景色*/
        cleardevice();
        circle(320, 240, 20+i*10);          /*画半径不同的圆*/
        delay(20000);
    }
    closegraph();
    return 0;
}
```

调试运行程序

程序运行结果如下：左图是在黑色背景上开始画不同颜色的同心圆，右图是背景切换成各种颜色，在中心某一时刻画一个圆的效果。





小讲堂

2．屏幕颜色的设置和清屏函数

对于图形模式的屏幕颜色设置，同样分为背景色的设置和前景色的设置。在 Turbo C 中分别用两个函数设置背景色函数 setbkcolor()和设置作图色函数 setcolor()。

（1）设置背景色函数 setbkcolor()的调用格式为：

```
void  far  setbkcolor(int  color);
```

（2）设置作图色函数 setcolor()的调用格式为：

```
void  far  setcolor(int  color);
```

其中 color 为图形方式下颜色的规定数值，对 EGA、VGA 显示器适配器，有关颜色的符号常数及数值如表 11-3 所示。

表 11-3　有关屏幕颜色的符号常数表

符号常数	数值	含义	符号常数	数值	含义
BLACK	0	黑色	DARKGRAY	8	深灰
BLUE	1	蓝色	LIGHTBLUE	9	深蓝
GREEN	2	绿色	LIGHTGREEN	10	淡绿
CYAN	3	青色	LIGHTCYAN	11	淡青
RED	4	红色	LIGHTRED	12	淡红
MAGENTA	5	洋红	LIGHTMAGENTA	13	淡洋红
BROWN	6	棕色	YELLOW	14	黄色
LIGHTGRAY	7	淡灰	WHITE	15	白色

对于 CGA 适配器，背景色可以为表 11-3 中 16 种颜色的一种，但前景色依赖于不同的调色板。共有 4 种调色板，每种调色板上有 4 种颜色可供选择。CGA 调色板与颜色值表如表 11-4 所示。

表 11-4　CGA 调色板与颜色值表

调　色　板		颜　色　值			
符号常数	数值	0	1	2	3
C0	0	背景	绿	红	黄
C1	1	背景	青	洋红	白
C2	2	背景	淡绿	淡红	黄
C3	3	背景	淡青	淡洋	红白

（3）清除图形屏幕内容使用清屏函数 cleardevice()，其调用格式如下：

```
void far cleardevice(void);
```

（4）获得当前颜色的几个函数。Turbo C 也提供了几个获得当前颜色设置情况的函数。

① 返回当前背景颜色函数 getbkcolor()。

函数原型如下：

```
int far getbkcolor(void);
```

函数功能：返回当前背景颜色值。

② 返回当前作图颜色值函数 getcolor()。

函数原型如下：

```
int far getcolor(void);
```

函数功能：返回当前作图颜色值。

③ 返回最多可用的颜色值函数 getmaxcolor()。

函数原型如下：

```
int far getmaxcolor(void);
```

函数功能：返回最多可用的颜色值。

三、基本画图函数

【思考题 11-6】 分析以下程序的运行结果，学习怎样使用基本画图函数。

程序分析

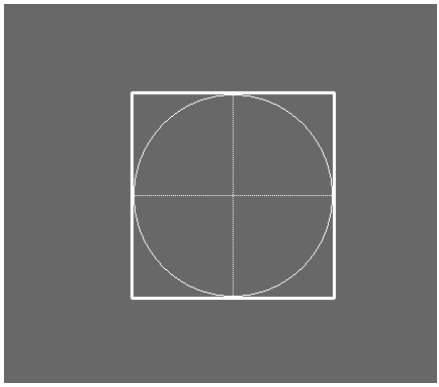
首先初始化图形界面，设置图形背景颜色为蓝色，当前颜色为绿色，然后以(320,240)为中心点，以 98 为半径画一个绿色的圆。再设置线的宽度为 3 点宽的实线。在圆的外面画一个线宽为 3 的矩形。再设置颜色为白色，设置 1 点宽的用户自定义线（虚线），在矩形中心画两条直线，正好将矩形分成四个相同小矩阵。

编写程序代码

```
#include  <stdlib.h>
#include  <graphics.h>
int main()
{   int  gdriver,gmode,i;
    gdriver=DETECT;
    initgraph(&gdriver, &gmode, "c:\\caic\\bgi");
    setbkcolor(BLUE);              /*设置背景色为蓝色*/
    cleardevice();
    setcolor(GREEN);               /*设置当前颜色为绿色*/
    circle(320, 240, 98);
    setlinestyle(0, 0, 3);         /*设置 3 点宽实线*/
    setcolor(2);                   /*设置当前颜色为绿色*/
    rectangle(220, 140, 420, 340);
    setcolor(WHITE);
    setlinestyle(4, 0xaaaa, 1);  /*设置 1 点宽用户定义线*/
    line(220, 240, 420, 240);
    line(320, 140, 320, 340);
    getch();
    closegraph();
    return 0;
}
```

调试运行程序

程序运行结果如下：



小讲堂

基本图形函数包括画点、线以及其他一些基本图形的函数。下面对这些函数进行介绍。

1．画点

1）画点函数

（1） 画点函数 putpixel()。函数原型如下：

```
void far putpixel(int x, int y, int color);
```

函数功能：该函数表示有指定的像元画一个 color 所指定颜色的点。对于颜色 color 的值可从表 11-3 中获得，而 x, y 是指图形像元的坐标。

在图形模式下，是按像元来定义坐标的。对 VGA 适配器，它的最高分辨率为 640×480，其中 640 为整个屏幕从左到右所有像元的个数，480 为整个屏幕从上到下所有像元的个数。屏幕的左上角坐标为(0,0)，右下角坐标为(639, 479)，水平方向从左到右为 x 轴正向，垂直方向从上到下为 y 轴正向。Turbo C 的图形函数都是相对于图形屏幕坐标，即像元来说的。

（2）获得当前点(x, y)的颜色值函数 getpixel。函数原型如下：

```
int  far  getpixel(int x, int y);
```

函数功能：获得当前点(x, y)的颜色值。

2）有关坐标位置的函数

（1）返回 x 轴的最大值函数 getmaxx()。函数原型如下：

```
int  far  getmaxx(void);
```

函数功能：返回 x 轴的最大值。

（2）返回 y 轴的最大值函数 getmaxy()。函数原型如下：

```
int  far  getmaxy(void);
```

函数功能：返回 y 轴的最大值。

（3）返回游标在 x 轴的位置函数 getx()。函数原型如下：

```
int  far  getx(void);
```

函数功能：返回游标在 x 轴的位置。

（4）返回游标在 y 轴的位置函数 gety()。函数原型如下：

```
void  far  gety(void);
```

函数功能：返回游标在 y 轴的位置。

（5）移动游标且画点函数 moveto()。函数原型如下：

```
void  far  moveto(int  x, int  y);
```

函数功能：移动游标到(x, y)点，在移动过程中也画点。

（6）移动游标位置且不画点函数 moverel()。函数原型如下：

```
void far moverel(int dx, int dy);
```

函数功能：将游标从当前位置(x,y)移动到(x+dx, y+dy)的位置，移动过程中不画点。



2．画线

1）画线函数

（1）画直线函数 line()。函数原型如下：

```
void far line(int x0, int y0, int x1, int y1);
```

函数功能：画一条从点(x0,y0)到(x1,y1)的直线。

（2）画圆函数 circle()。函数原型如下：

```
void far circle(int x, int y, int radius);
```

函数功能：以(x, y)为圆心，radius 为半径，画一个圆。

（3）画圆弧函数 arc()。函数原型如下：

```
void far arc(int x, int y, int stangle, int endangle,int radius);
```

函数功能：以(x,y)为圆心，radius 为半径，从 stangle 开始到 endangle 结束（用度表示）画一段圆弧线。

在 Turbo C 中规定 x 轴正向为 0°，逆时针方向旋转一周，依次为 90°、180°、270°和 360°（其他有关函数也按此规定，不再重述）。

（4）画椭圆函数 ellipse()。函数原型如下：

```
void ellipse(int x, int y, int stangle, int endangle,int xradius,int yradius);
```

函数功能：以(x, y)为中心，xradius、yradius 分别为 x 轴和 y 轴半径，从角 stangle 开始到 endangle 结束画一段椭圆线，当 stangle=0，endangle=360 时， 画出一个完整的椭圆。

（5）画矩形函数 rectangle()。函数原型如下：

```
void far rectangle(int x1, int y1, int x2, inty2);
```

函数功能：以(x1, y1)为左上角，(x2, y2)为右下角画一个矩形框。

（6）画多边形函数 drawpoly()。函数原型如下：

```
void far drawpoly(int numpoints, int far *polypoints);
```

函数功能：画一个顶点数为 numpoints，各顶点坐标由 polypoints 给出的多边形。polypoints 整型数组必须至少有 2 倍顶点数个元素。每一个顶点的坐标都定义为(x,y)，并且 x 在前。需要注意的是当画一个封闭的多边形时，numpoints 的值取实际多边形的顶点数加 1，并且数组 polypoints 中第一个和最后一个点的坐标相同。

2）设定线型函数

在没有对线的特性进行设定之前，Turbo C 用其默认值，即 1 点宽的实线，但 Turbo C 也提供了可以改变线型的函数。

线型包括：宽度和形状。其中宽度只有两种选择：1 点宽和 3 点宽。而线的形状则有 5 种。

（1）线型的设置函数 setlinestyle()。函数原型如下：

```
void far setlinestyle(int  linestyle,unsigned  upattern,int thickness);
```

函数功能：该函数用来设置线的有关信息，包括线的形状和线的宽度等。其中 linestyle 是线的形状的规定，如表 11-5 所示。

表 11-5　有关线的形状(linestyle)

符 号 常 数	数　值	含　义
SOLID_LINE	0	实线
DOTTED_LINE	1	点线
CENTER_LINE	2	中心线
DASHED_LINE	3	点画线
USERBIT_LINE	4	用户定义线

thickness 是线的宽度，如表 11-6 所示。

表 11-6　有关线宽(thickness)

符 号 常 数	数　值	含　义
NORM_WIDTH	1	1 点宽
THIC_WIDTH	3	3 点宽

对于 upattern，只有 linestyle 选 USERBIT_LINE 时才有意义（选其他线型，uppattern 取 0 即可）。此时 uppattern 的 16 位二进制数的每一位代表一个像元，如果 16 位二进制数的某一位为 1，则对应该像元打开，否则该像元关闭。

（2）返回线信息函数 getlinesettings()。函数原型如下：

```
void far getlinesettings(struct linesettingstypefar *lineinfo);
```

函数功能：该函数将有关线的信息存放到由 lineinfo 指向的结构中，其中 linesettingstype 的结构如下：

```
struct  linesettingstype
{  int  linestyle;
   unsigned  upattern;
   int  thickness;
}
```

例如，下面两条语句程序可以读出当前线的特性：

```
struct  linesettingstype  *info;
getlinesettings(info);
```

（3）画线方式设置函数 setwritemode()。函数原型如下：

```
void far setwritemode(int mode);
```

函数功能：该函数规定了画线的方式。如果 mode=0，则表示画线时将所画位置的原来信息覆盖了（这是 Turbo C 的默认方式）。如果 mode=1，则表示画线时用现在特性的线与所画之处原有的线进行异或（XOR）操作，实际上画出的线是原有线与现在规定的线进行异或后的结果。因此，当线的特性不变，进行两次画线操作相当于没有画线。

练一练

【练习 11-2】 要求画出半径逐渐增大的同心圆弧。

解：首先屏幕初始化，然后使用循环语句，多次以点(320,240)为圆心，以 b（b 值为 10～140 之间，每次增加 10）为半径，从 90° 到 180° 画多个 1/4 的圆弧。

```
#include <graphics.h>
void main()
{  int b;
   int gdriver=DETECT,gmode;
   initgraph(&gdriver,&gmode,"c:\\tc");
   cleardevice();
   printf("\n\n\n  This program shows the arc graph.\n");
   for(b=10;b<=140;b+=10)
      arc(320,240,0,150,b);
   getch();
   closegraph();
}
```

程序运行结果如下：



本讲小结

本讲主要介绍了图形模式的初始化，屏幕颜色的设置和清屏函数，以及几个基本画图函数。注意这些函数在使用过程中的参数设置。

想一想

本讲我们讲了几个基本的画图函数，例如画圆函数。如果想画一个圆，又让这个圆中区域填充成蓝色，应该怎么做呢？

第三十讲　图形处理函数（二）

一、基本图形的填充及填充方式的设定

【思考题 11-7】　分析以下程序的运行结果，注意各图形的填充方法。



程序分析

该程序初始化后，设置界面背景为蓝色，在界面上画了 4 个图形：矩形，长方体，扇形和椭圆扇形。并用各种红色填充模式进行填充，最后用黄色画出各图形的边框。



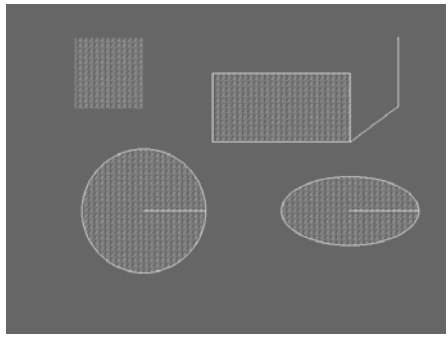
编写程序代码

```
#include <graphics.h>
main()
{   char str[8]={10,20,30,40,50,60,70,80}; /*用户定义图模*/
    int gdriver,gmode,i;
    struct fillsettingstype save;   /*定义一个用来存储填充信息的结构变量*/
    gdriver=DETECT;
    initgraph(&gdriver,&gmode,"c:\\tc");
    setbkcolor(BLUE);
    cleardevice();
    for(i=0;i<13;i++)
    {   setcolor(i+3);
        setfillstyle(i,2+i);                /*设置填充类型 */
        bar(100,150,200,50);                /*画矩形并填充*/
        bar3d(300,100,500,200,70,1);        /*画长方体并填充*/
        pieslice(200, 300, 90, 180, 90);    /*画扇形并填充*/
        sector(500,300,180,270,200,100);    /*画椭圆扇形并填充*/
        delay(1000);                        /*延时 1 秒*/
    }
    cleardevice();
    setcolor(14);
    setfillpattern(str,RED);
    bar(100,150,200,50);
    bar3d(300,100,500,200,70,0);
    pieslice(200,300,0,360,90);
    sector(500,300,0,360,100,50);
    getch();
    getfillsettings(&save);       /*获得用户定义的填充模式信息*/
    closegraph();
    clrscr();
    printf("The pattern is %d, The color of filling is %d", save.pattern,
            save.color);          /*输出目前填充图模和颜色值*/
    getch();
}
```



调试运行程序

程序运行结果如下：



小讲堂

1．基本图形的填充

填充就是用规定的颜色和图模填满一个封闭图形。

先画轮廓再填充。Turbo C 提供了一些先画出基本图形轮廓，再按规定图模和颜色填充整个封闭图形的函数。在没有改变填充方式时，Turbo C 以默认方式填充。

（1）矩形填充函数 bar()。函数原型如下：

```
void far bar(int x1, int y1, int x2, int y2);
```

函数功能：确定一个以(x1,y1)为左上角，(x2,y2)为右下角的矩形窗口，再按规定图模和颜色填充。

说明：此函数不画出边框，所以填充色为边框。

（2）三维长方体填充函数 bar3d()。函数原型如下：

```
void far bar3d(int x1, int y1, int x2, int y2,int depth,int topflag);
```

函数功能：当 topflag 为非 0 时，画出一个三维的长方体。当 topflag 为 0 时，三维图形不封顶，实际上很少这样使用。

说明：bar3d()函数中，长方体第三维的方向不随任何参数而变，即始终为 45° 的方向。

（3）扇形填充函数 pieslice()。函数原型如下：

```
void far pieslice(int x,int y,int stangle,int endangle,int radius);
```

函数功能：画一个以(x, y)为圆心，radius 为半径，stangle 为起始角度，endangle 为终止角度的扇形，再按规定方式填充。当 stangle=0,endangle=360 时变成一个实心圆，并在圆内从圆点沿 x 轴正向画一条半径。

（4）椭圆填充函数 sector()。函数原型如下：

```
void far sector(int x, int y,int stanle,intendangle,int xradius, int yradius);
```

函数功能：画一个以(x, y)为圆心分别以 xradius、 yradius 为 x 轴和 y 轴半径，stangle 为起始角，endangle 为终止角的椭圆扇形，再按规定方式填充。

2．设定填充方式

Turbo C 有 4 个与填充方式有关的函数，下面分别介绍。

（1）设置填充模式和颜色函数 setfillstyle()。该函数的调用格式为：

```
void  far  setfillstyle(int  pattern, int  color);
```

该函数功能是用来设置当前填充模式和填充颜色，以便用于填充一个指定的封闭区域。color 的值是当前屏幕图形模式时颜色的有效值，pattern 的值及与其等价的符号常数和含义如表 11-7 所示。

表 11-7　关于填充式样 pattern 的规定

符 号 常 数	数　值	含　义
EMPTY_FILL	0	以背景颜色填充
SOLID_FILL	1	以实体填充
LINE_FILL	2	以直线填充
LTSLASH_FILL	3	以斜线填充（阴影线）
SLASH_FILL	4	以粗斜线填充（粗阴影线）
BKSLASH_FILL	5	以粗反斜线填充（粗阴影线）
LTBKSLASH_FILL	6	以反斜线填充（阴影线）
HATCH_FILL	7	以直方网格填充
XHATCH_FILL	8	以斜网格填充
INTTERLEAVE_FILL	9	以间隔点填充
WIDE_DOT_FILL	10	以稀疏点填充
CLOSE_DOS_FILL	11	以密集点填充
USER_FILL	12	以用户定义式样填充

除 USER_FILL（用户定义填充式样）以外，其他填充式样均可由 setfillstyle() 函数设置。当选用 USER_FILL 时，该函数对填充图模和颜色不做任何改变。之所以定义 USER_FILL 主要因为在获得有关填充信息时用到此项。

（2）设置填充图模颜色函数 setfillpattern()。该函数的调用格式为：

```
void far setfillpattern(char * upattern,int color);
```

该函数的功能是设置用户定义的填充图模的颜色以供对封闭图形填充。其中 upattern 是一个指向 8 字节的指针。这 8 字节定义了 8×8 点阵的图形。每个字节的 8 位二进制数表示水平 8 点，8 个字节表示 8 行，然后以此为模型向个封闭区域填充。

（3）存储填充图模函数 getfillpattern()。函数原型如下：

```
void far getfillpattern(char * upattern);
```

函数功能：该函数将用户定义的填充图模存入 upattern 指针指向的内存区域。

（4）返回填充图模信息函数 getfillsetings()。函数原型如下：

```
void far getfillsetings(struct fillsettingstype  far * fillinfo);
```

该函数获得现行图模的颜色并将其存入结构指针变量 fillinfo 中。其中 fillsettingstype 结构定义如下：

```
struct  fillsettingstype
{   int  pattern;    /* 现行填充模式 * /
    int  color;      /* 现行填充颜色 * /
};
```

二、任意封闭图形的填充

【思考题 11-8】 本例是有关 floodfill()函数的用法。该程序填充了使用函数 bar3d()所画长方体中其他两个未填充的面。



程序分析

该程序初始化后，首先设置界面背景为蓝色，其次设置当前颜色为淡红色，设置线型为实线，线宽为 3 点宽。再设置填充为实填充，填充颜色为黄色。最后画出长方体，并将未填充的两个面以黄色填充。再画一个小矩形（边框为淡红色），最后对矩形用黄色进行填充。



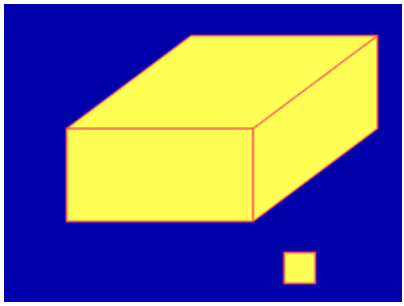
编写程序代码

```
#include<stdlib.h>
#include<graphics.h>
main()
{   int  gdriver,  gmode;
    struct  fillsettingstype save;
    gdriver=DETECT;
    initgraph(&gdriver, &gmode, "c:\\caic\\bgi");
    setbkcolor(BLUE);                  /*设置背景颜色为蓝色*/
    cleardevice();
    setcolor(LIGHTRED);                /*设置当前颜色为淡红色*/
    setlinestyle(0,0,3);               /*设置线型为实线，线宽为 3 点宽*/
    setfillstyle(1,14);                /*设置填充方式*/
    bar3d(100,200,400,350,200,1);      /*画长方体并填充*/
    floodfill(450,300,LIGHTRED);
    /*填充长方体另外两个面*/
    floodfill(250,150, LIGHTRED);
    rectangle(450,400,500,450);        /*画一矩形*/
    floodfill(470,420, LIGHTRED);      /*填充矩形*/
    getch();
    closegraph();
}
```



调试运行程序

程序运行结果如下：



小讲堂

3．任意封闭图形的填充函数

Turbo C 提供了一个可对任意封闭图形填充的函数，其调用格式如下：

```
void far floodfill(int x, int y, int border);
```

该函数的功能是用规定的颜色和图模填满整个封闭图形。其中 x,y 为封闭图形内的任意一个 border 为边界的颜色，也就是封闭图形轮廓的颜色。

注意

（1）如果 x 或 y 取在边界上，则不进行填充。

（2）如果不是封闭图形则填充会从没有封闭的地方溢出去，填满其他地方。

（3）如果 x 或 y 在图形外面，则填充封闭图形外的屏幕区域。

（4）由 border 指定的颜色值必须与图形轮廓的颜色值相同，但填充色可选任意颜色。

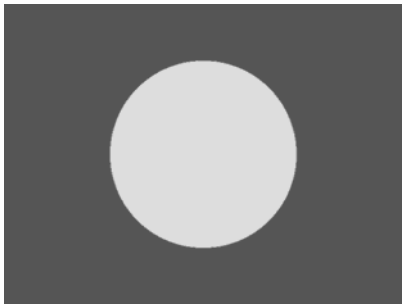
练一练

【练习 11-3】 编写一个程序，实现用红色画一个圆形，并用绿色对该圆进行填充。

解：首先初始化图形界面，设置当前颜色为红色，然后用画圆函数，以点(320,240)为圆心，以 150 为半径画一个圆。然后再设置填充模式：颜色为绿色，填充模式为实填充。最后将圆的边框画成红色。源程序如下：

```
#include  <graphics.h>
main()
{   int  b;
    int  gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode,"c:\\tc");
    cleardevice();
    setcolor(RED);
    circle(320,240,150);
    setfillstyle(SOLID_FILL,GREEN);
    floodfill(320,240,RED);
    getch();
    closegraph();
}
```

程序运行结果如下：



本讲小结

本讲主要介绍了基本图形的填充，填充方式的设定以及任意封闭图形的填充。

想一想

图形的填充函数有哪些？各自的参数都代表什么含义？

第三十一讲　图形操作函数

学习目标

- 掌握图形窗口操作函数;
- 掌握图形模式下的字符的各种函数。

一、图形窗口操作

【思考题 11-9】 编写程序模拟两个小球动态碰撞的过程。

程序分析

该程序初始化后，设置界面背景为蓝色，设置当前颜色为淡红色，线型为实线，线宽为 1 点宽。然后再设置填充为实填充，填充颜色为黄色，画出小球。通过循环地将屏幕图形存储到内存，再从内存中取出，达到动画的效果。

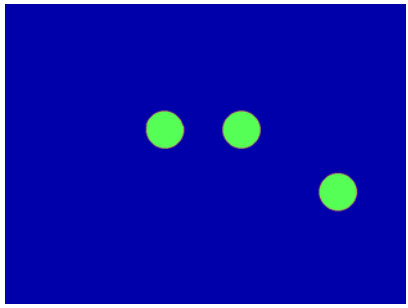
编写程序代码

```
#include  <malloc.h>
#include  <stdio.h>
#include  <graphics.h>
main()
{   int  i, gdriver, gmode, size;
    void  *buf;
    gdriver=DETECT;
    initgraph(&gdriver, &gmode, "c:\\tc");
    setbkcolor(BLUE);
    cleardevice();
    setcolor(LIGHTRED);
    setlinestyle(0,0,1);
    setfillstyle(1, 10);
    circle(100, 200, 30);
    floodfill(100, 200, 12);
```

```
    size=imagesize(69, 169, 131, 231);
    buf=malloc(size);
    if(!buf)
       return -1;
    getimage(69, 169, 131, 231,buf);
    putimage(500, 269, buf, COPY_PUT);
    for(i=0; i<185; i++)
    {   putimage(70+i, 170, buf, COPY_PUT);
       putimage(500-i, 170, buf, COPY_PUT);
    }
    for(i=0;i<185; i++)
    {   putimage(255-i, 170, buf, COPY_PUT);
       putimage(315+i, 170, buf, COPY_PUT);
    }
    getch();
    closegraph();
}
```

调试运行程序

程序运行结果如下：



小讲堂

1．图形操作函数

如同字符方式下可以设定屏幕窗口一样，图形方式下也可以在屏幕上某一区域设定窗口，只是设定的为图形窗口而已，其后的有关图形操作都将以这个窗口的左上角(0, 0)作为坐标原点，而且可为通过设置使窗口之外的区域为不可接触。这样，所有的图形操作就被限定在窗口内进行。

（1）设定图形窗口函数 setviewport()，其调用格式如下：

```
void  far  setviewport(int  xl, int  yl, int  x2, int  y2, int  clipflag);
```

该函数的功能是：设定一个以(x1,y1)像元为左上角，(x2,y2)像元为右下角的图形窗口，其中 x1,y1,x2,y2 是相对于整个屏幕的坐标。若 clipflag 为非 0，则设定的图形以外部分不可接触；若 clipflag 为 0，则图形窗口以外可以接触。

（2）清除现行图形窗口的内容函数 clearviewport()，其调用格式如下：

```
void  far  clearviewport(void);
```

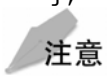
该函数的功能是清除现行图形窗口中的内容。

（3）返回现行窗口信息函数 getviewsettings()，其调用格式如下：

```
void far getviewsettings(struct viewporttype far *viewport);
```

该函数的功能是获得关于现行窗口的信息，并将其存于 viewporttype 定义的结构变量 viewport 中，其中 viewporttype 的结构说明如下：

```
struct viewporttype
{ int left, top, right, bottom;
int cliplag;
};
```



（1）窗口颜色的设置与前面讲过的屏幕颜色设置相同，但屏幕背景色和窗口背景色只能是一种颜色，如果窗口背景色改变，则整个屏幕的背景色也将改变，这与字符窗口不同。

（2）可以在同一个屏幕上设置多个窗口，但只能有一个现行窗口工作，要对其他窗口操作，将定义那个窗口的 setviewport()函数再用一次即可。

（3）前面讲过图形屏幕操作的函数均适合于对窗口的操作。



2．屏幕操作函数

除了清屏函数以外，关于屏幕操作还有以下几个函数。

（1）为图形输出选择激活页函数 setactivepage()。函数原型如下：

```
void far setactivepage(int pagenum);
```

函数功能：setctivepage()函数是为图形输出选择激活页。所谓激活页，是指后续图形的输出被写到函数选定的 pagenum 页面，该页面并不一定可见。

（2）使指定页变成可见页函数 setvisualpage()。函数原型如下：

```
void far setvisualpage(int pagenum);
```

函数功能：setvisualpage()函数才使 pagenum 所指定的页面变成可见页，页面从 0 开始（Turbo C 默认页）。

这两个函数只用于EGA、VGA 以及HERCULES图形适配器。如果先用setactivepage() 函数在不同页面上画出一幅幅图像，再用 setvisualpage() 函数交替显示，就可以实现一些动画的效果。

（3）屏幕图像存取函数，有以下 3 个函数，其原型分别为：

```
void far getimage(int xl,int yl, int x2,int y2,void far *mapbuf);
void far putimge(int x,int,y,void * mapbuf, int op);
unsined far imagesize(int xl,int yl,int x2,int y2);
```

这 3 个函数用于将屏幕上的图像复制到内存，然后将内存中的图像送回到屏幕上。首先通过函数 imagesize() 测试要保存左上角为(x1,y1)，右下角为(x2,y2)的图形屏幕区域内的全部

内容需多少字节，然后再给 mapbuf 分配一个所测数字节内存空间的指针。通过调用 getimage()函数就可将该区域内的图像保存在内存中，需要时可用 putimage()函数将该图像输出到左上角为点(x, y)的位置上，其中 putimage()函数中的参数 op 规定如何释放内存中图像。 关于这个参数的定义如表 11-8 所示。

表 11-8　putimage()函数中的 op 值

符 号 常 数	数　值	含　义
COPY_PUT	0	复制
XOR_PUT	1	与屏幕图像异或后复制
OR_PUT	2	与屏幕图像或后复制
AND_PUT	3	与屏幕图像与后复制
NOT_PUT	4	复制反像的图形

对于 imagesize()函数，只能返回字节数小于 64KB 的图像区域，否则将会出错，出错时返回−1。

本讲介绍的这些函数在图像动画处理、菜单设计技巧中非常有用。

二、图形模式下的字符

【思考题 11-10】　分析以下程序，注意其中的图形模式的字符的显示方法。



程序分析

该程序初始化后，设置界面背景为蓝色，然后定义了一个图形窗口，并以绿色进行填充。再设置颜色为黄色，画这个窗口的边框。设置颜色为淡红色，设置字符的模式为三重笔画字体，水平放大 8 倍，在屏幕上点(20,20)输出字符串"Good Better"。设置颜色为白色，设置字符的模式为无衬笔画字体，水平放大 5 倍，在点(120,120) 输出显示字符串"Good Better"。设置颜色为黄色，设置字符模式为小号笔画字体，水平放大 2 倍，在点(30,200)输出字符串"Your score is 620"。再设置字符颜色为蓝色，设置字符的模式为黑体笔画字，在点(70,240)输出字符串"Your score is 620"。



编写程序代码

```
#include<graphics.h>
#include<stdio.h>
int main()
{   int i, gdriver, gmode;
    char s[30];
    gdriver=DETECT;
    initgraph(&gdriver, &gmode, "c:\\caic\\bgi");
    setbkcolor(BLUE);
    cleardevice();
    setviewport(100, 100, 540, 380, 1);  /*定义一个图形窗口*/
    setfillstyle(1, 2);                  /*绿色以实填充*/
    setcolor(YELLOW);
    rectangle(0, 0, 439, 279);
    floodfill(50, 50, 14);
    setcolor(12);
```

```
    settextstyle(1, 0, 8);                /*三重笔画字体，水平放大 8 倍*/
    outtextxy(20, 20, "Good Better");
    setcolor(15);
    settextstyle(3, 0, 5);                /*无衬笔画字体，水平放大 5 倍*/
    outtextxy(120, 120, "Good Better");
    setcolor(14);
    settextstyle(2, 0, 8);
    i=620;
    sprintf(s, "Your score is %d", i);    /*将数字转化为字符串*/
    outtextxy(30, 200, s);                /*指定位置输出字符串*/
    setcolor(1);
    settextstyle(4, 0, 3);
    outtextxy(70, 240, s);
    getch();
    closegraph();
    return 0;
}
```



调试运行程序

程序运行结果如下：





在图形模式下，只能用标准输出函数，如 printf()、puts()、putchar()函数输出文本到屏幕。除此之外，其他输出函数（如窗口输出函数）不能使用，即使可以输出的标准函数，也只以前景色为白色，按 80 列、25 行的文本方式输出。

Turbo C 2.0 提供了一些专门用于在图形显示模式下的文本输出函数。

3．文本输出函数

（1）当前位置输出字符函数 outtext()。函数原型如下：

```
void far outtext(char far *textstring);
```

函数功能：该函数输出字符串指针 textstring 所指的文本在现行位置。

（2）指定位置输出字符函数 outtextxy()。函数原型如下：

```
void far outtextxy(int x, int y, char far *textstring);
```

函数功能：该函数输出字符串指针 textstring 所指的文本在规定的(x,y)位置，其中(x,y)为像元坐标。

说明：这两个函数都是输出字符串，但经常会遇到输出数值或其他类型的数据，此时就必须使用格式化输出函数 sprintf()。

（3）按格式化规定的内容写字符串函数 sprintf()。函数原型如下：

```
int   sprintf(char   *str, char   *format, variable-list);
```

函数功能：它与 printf()函数不同之处是将按格式化规定的内容写入 str 指向的字符串中，返回值等于写入的字符个数。例如：

sprintf(s, "your TOEFL score is %d", mark);

这里 s 应是字符串指针或数组，mark 为整型变量。



4．文本字体、字型和输出方式的设置

有关图形方式下的文本输出函数，可以通过 setcolor()函数设置输出文本的颜色。另外，也可以改变文本字体大小以及选择是水平方向输出还是垂直方向输出。

1）定位输出字符串函数 settextjustify()

函数原型如下：

```
void far settexjustify(int horiz, int vert);
```

函数功能：该函数用于定位输出字符串。

对使用 outtextxy(int x, int y, char far *str textstring)函数所输出的字符串，其中哪个点对应于定位坐标(x,y)在 Turbo C 2.0 中是有规定的。如果把一个字符串看成一个长方形的图形，在水平方向显示时，字符串长方形按垂直方向可分为顶部、中部和底部 3 个位置，水平方向可分为左、中、右 3 个位置，两者结合就有 9 个位置。

settextjustify()函数的第一个参数 horiz 指出水平方向 3 个位置中的一个，第二个参数 vert 指出垂直方向 3 个位置中的一个，二者就确定了其中一个位置。当规定了这个位置后，用 outtextxy() 函数输出字符串时，字符串长方形的这个规定位置就对准函数中的(x,y)位置。而对用 uttext()函数输出字符串时，这个规定的位置就位于现行游标的位置。

有关参数 horiz 和 vert 的取值如表 11-9 所示。

表 11-9　参数 horiz 和 vert 的取值

符 号 常 数	数　　值	用　　于
LEFT_TEXT	0	水平
RIGHT_TEXT	2	水平
BOTTOM_TEXT	0	垂直
TOP_TEXT	2	垂直
CENTER_TEXT	1	水平或垂直

2）设置字符模式函数 settextstyle()

在图形方式下，BGI 提供了两种输出字符的方式，一种是位映像字符（或称点阵字符）；另一种是笔画字体（或称矢量字符）。位映像字符为缺省方式，即在一般情况下输出字符时，都是以位映像字符显示的。

笔画字符不是以位模式表示的，每个字符被定义为一系列的线段或笔画的组合，笔画字体

可以灵活地改变其大小，而且不会降低其分辨率。系统提供了 4 种不同的笔画字体，即小号字体、3 倍字体、无衬线字体和黑体。每种笔画字体都存放在独立的字体文件中，文件扩展名为.chr，一般情况下安装在与 BGI 相同目录下。为了使用笔画字体，必须装入相应的字体文件。

设置字符模式函数 settextstyle()的调用格式为：

```
void far settextstyle(int font, int direction,int charsize);
```

该函数用来设置输出字符的字形（由 font 确定）、输出方向（由 direction 确定）和字符大小（由 charsize 确定）等特性。Turbo C 2.0 对函数中各个参数的规定如表 11-10 至表 11-12 所示。

表 11-10　font 的取值

符 号 常 数	数　　值	含　　义
DEFAULT_FONT	0	8×8 点阵字（缺省值）
TRIPLEX_FONT	1	三倍笔画字体
SMALL_FONT	2	小号笔画字体
SANSSERIF_FONT	3	无衬线笔画字体
GOTHIC_FONT	4	黑体笔画字

表 11-11　direction 的取值

符 号 常 数	数　　值	含　　义
HORIZ_DIR	0	从左到右
VERT_DIR	1	从底到顶

表 11-12　charsize 的取值

符号常数或数值	含　　义
1	8×8 点阵
2	16×16 点阵
3	24×24 点阵
4	32×32 点阵
5	40×40 点阵
6	48×48 点阵
7	56×56 点阵
8	64×64 点阵
9	72×72 点阵
10	80×80 点阵
USER_CHAR_SIZE=0	用户定义的字符大小



5．字符大小的设置函数settextstyle()

前面介绍的 settextstyle()函数，可以设定图形方式下输出文本字符的字体和大小。但对于

笔画型字体（除 8×8 点阵字以个的字体），只能在水平和垂直方向以相同的放大倍数放大。

为此 Turbo C 2.0 又提供了另外一个 setusercharsize()函数，对笔画字体可以分别设置水平和垂直方向的放大倍数。该函数的调用格式为：

```
void far setusercharsize(int mulx, int divx, int muly, int divy);
```

该函数用来设置笔画型字和放大系数，它只有在 settextstyle()函数中的 charsize 为 0（USER_CHAR_SIZE）时才起作用，并且字体为函数 settextstyle()规定的字体。调用函数 setusercharsize()后，每个显示在屏幕上的字符都以其默认大小乘以 mulx/divx 为输出字符宽，乘以 muly/divy 为输出字符高。

练一练

【练习 11-4】 设计一个由小逐渐变大的字符串"Hello!"，使其显示在屏幕中央。

解：使用字符大小的设置函数 settextstyle()，可以改变字符的大小和字体，对字符进行各种设置。

```
#include <graphics.h>
#include <dos.h>
void main()
{   int i,t,x=300,y=50;
    int gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode,"c:\\tc");
    setbkcolor(9);
    setcolor(4);
    printf("Please input delay time (1-10): ");
    scanf("%d",&t);
    outtextxy(80,20,"This program show the Text in graphic mode.");
    for(i=1;i<=10;i++)
    {  x=x-15;
       y=y+15;
       settextstyle(1,0,i);              /*cleardevice();*/
       outtextxy(x,y,"Hello!");
       delay(1000*t);
    }
    settextstyle(1,0,1);
    outtextxy(80,420,"Press any key to quit...");
    getch();
    closegraph();
}
```

程序运行结果为：



本讲小结

本讲主要介绍了图形模式下的字符字体、字型和输出方式的设置和用户对文本字符大小的设置等函数。

想一想

怎样能用好图形函数？如果想改变字符的大小，应该使用什么函数？

本章小结

本章主要介绍了字符屏幕处理函数和图形处理函数。字符屏幕处理函数主要介绍了字符窗口的定义、颜色设置、字符窗口的输入/输出函数。在图形处理函数中，介绍了图形模式的初始化、屏幕颜色设置、清屏函数、画点、画线函数、封闭图形的填充函数、图形屏幕操作函数、图形模式下的字符输出函数等。

课后习题十一

编程题

1. 设计一个程序，制作多变的椭圆。
2. 设计一个程序，制作统计用的饼形图。
3. 设计一个程序，制作在屏幕上飞舞的多彩曲线。

第十二章　上机实训

第三十二讲　学生成绩管理系统

一、问题描述

设计一个学生管理系统。该系统可以建立学生信息表，向表中插入新记录，删除某记录，或在表中查找某特定的学生信息。可输入一组学生成绩，包括每名学生的学号，姓名，语文、数学和英语成绩，建立学生成绩信息表。可加入新学生信息到学生成绩表中，或删除表中的某个位置的学生，或在表中查找某位置的学生并将该学生的信息显示出来。在主函数中设计一个菜单，将要进行的操作用 0～4 来表示，允许用户输入相应的数字来进行反复操作。

二、数据结构

其主要的数据类型为带头结点的单链表：

```
typedef  struct  node
{   int number;                    /*学生的学号*/
    char name[20];                 /*姓名*/
    int chinese,math,english;      /*语文、数学和英语成绩*/
    struct  node   *next;
}LinkList;
```

三、程序流程

1．函数之间的调用关系图

程序流程图如图 12-1 所示。

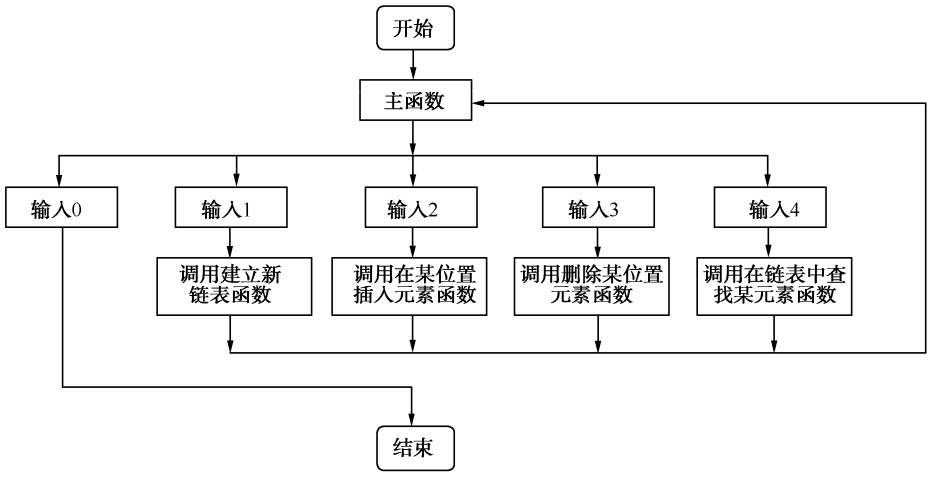


图 12-1　“学生成绩管理系统”程序流程图

2．主要函数

（1）初始化链表：LinkList　*InitList()。

（2）求链表长度：int　Length_List(LinkList *H)。

（3）建立链表函数：void CreateList(LinkList *H,int n)。

（4）表中元素定位函数：LinkList *Locate(LinkList *H,int num)。

（5）查找表中元素位置函数：LinkList *GetList(LinkList *H,int i)。

（6）在链表中插入新元素函数：int InsList(LinkList *p,LinkList x)。

（7）在给定位置 i 插入元素函数：int Ins_List(LinkList *H, int i, LinkList x)。

（8）删除链表中的某元素函数：int DelList(LinkList *p,LinkList *x)。

（9）删除链表中给定位置的元素函数：int Del_List(LinkList *H,int i,LinkList *x)。

（10）主函数中的菜单显示函数：out()。

（11）输出表中信息函数：void DisLinkList(LinkList *H)。

（12）主函数：main()。

四、完整程序

完整程序如下：

```
#include  "stdio.h"
#include  "string.h"
typedef  char  ElemType;
#define  OK     1                    /*操作成功返回 1*/
#define  ERROR  0                    /*错误为 0*/
#define  OVER   -1                   /*结束为-1*/
typedef  struct  node
{   int  number;                     /*学生学号*/
    char  name[20];                  /*学生姓名*/
    int  chinese,math,english;       /*学生语文、数学和英语成绩*/
    struct  node  *next;             /*指向下一结点的指针*/
}  LinkList;                         /*定义学生结点类型*/
/*********************初始化链表**********************/
LinkList  *InitList()
{   LinkList  *H;
    /*申请一块 LinkList 类型的存储单元的操作，并将其地址赋值给头指针变量 H*/
    H=(LinkList *)malloc(sizeof(LinkList)) ;
    H->next=NULL;
    return(H);                       /*头结点 H 指针域为空，表示空链表*/
}
/******************求链表长度************************/
int  Length_List(LinkList  *H)
{   LinkList  *p;
    int  j=0;
    p=H;                             /*p 指向链表的头结点*/
    while(p->next!=NULL)             /*判断 p 所指结点后面是否为空*/
    {   p=p->next;                   /*p 向所指结点的后面移动*/
        j++;                         /*计数器值增 1*/
    }
    return j;                        /*计数器 j 的值为表长度*/
}
```

```
/*****************建立链表函数***********************/
void  CreateList(LinkList  *H,int n)
{   int i;
    LinkList  *s,*last;
    char ch;
    last=H;               /*last 始终指向尾结点，开始时指向头结点*/
    for(i=1;i<=n;i++)     /*循环输入各学生成绩*/
    {   s=(LinkList *) malloc(sizeof(LinkList)); /*生成新结点*/
        printf("\n 请输入第%d 个学生的学号：",i);
        scanf("%d",&s->number);
        printf("\n 请输入第%d 个学生的姓名：",i);
        scanf("%s",s->name);
        printf("\n 请输入第%d 个学生的语文、数学和英语成绩：",i);
        scanf("%d%d%d",&s->chinese,&s->math,&s->english);
        s->next=NULL;                  /*将新结点的指针域为空*/
        last->next=s;                  /*将新结点插入表尾*/
        last=s;                        /*将 last 指针指向表尾*/
    }
}
/*********************表中元素定位函数******************/
LinkList  *Locate(LinkList  *H,int  num)
{   LinkList  *p;
    p=H->next;                        /*p 指向链表的第一个结点*/
    while(p!=NULL  &&  p->number!=num)
    p=p->next;

    return p;
}
/*****************查找表中元素位置函数******************/
LinkList *GetList(LinkList *H,int i)
{   LinkList  *p;
int j=0;
p=H;                                  /*p 指向链表的头结点*/
while(p->next!=NULL && j<i)
{   p=p->next;
j++;
}
if(j==i)                              /*判断与给定的序号是否相等*/
return  p;
else
return  NULL;
}
/*****************在链表中插入新元素函数*****************/
int InsList(LinkList *p,LinkList x)
{   LinkList  *s;
   s=(LinkList *)malloc(sizeof(LinkList));
   s->number=x.number;
   strcpy(s->name,x.name);
   s->chinese=x.chinese;
   s->math=x.math;
   s->english=x.english;              /*将数据放入新结点的数据域*/
   s->next=p->next;                   /*将新结点的指针域与 p 结点后面元素相连*/
```

```
   p->next=s;                          /*将新结点插入链表*/
   return OK;
}
/***************在给定位置 i 插入元素函数***************/
int  Ins_List(LinkList  *H, int  i, LinkList  x)
{  LinkList  *p;
   p=GetList(H, i-1);                  /*调用按序号查找第 i 个元素地址 p 函数*/
   if(p!=NULL)                         /*判断查找的元素地址 p 是否存在*/
   {  InsList(p,x);                    /*调用在已知结点 p 后插入结点算法函数*/
      return OK;
   }
   else
      return ERROR;
}
/*****************删除链表中的某元素函数*****************/
int  DelList(LinkList  *p,LinkList  *x)
{  LinkList  *s;
   s=p->next;                          /*s 为要删除结点*/
   x->number=s->number;
   strcpy(x->name,s->name);
   x->chinese=s->chinese;
   x->math=s->math;
   x->english=s->english;              /*将要删除的数据赋予指针变量 X */
   p->next=s->next;                    /*将 p 结点的指针域与 s 结点后面元素相连*/
   free(s);                            /*释放结点 s*/
   return OK;
}
/****************删除链表中给定位置的元素函数**************/
int  Del_List(LinkList  *H,int  i,LinkList  *x)
{  LinkList  *p;
   p=GetList(H, i-1);                 /*调用按序号查找第 i-1 个元素地址 p 函数*/
   if (p!=NULL && p->next!=NULL)      /*判断查找的元素地址 p 是否存在*/
   {  DelList(p,x);                   /*调用删除已知结点 p 之后结点函数*/
      return OK;
   }
   else
      return ERROR;
}
/*********************主函数中的菜单显示*****************/
out()
{  printf("\n*****************************************");
   printf("\n 请输入序号（0-4）选择要进行的操作:");
   printf("\n0------退出");
   printf("\n1------建立一个学生信息表");
   printf("\n2------插入一个学生信息");
   printf("\n3------删除一个学生信息");
   printf("\n4------查找一个学生是否在该学生信息表中");
   printf("\n*****************************************\n");
}
/*********************输出表中信息函数*****************/
void  DisLinkList(LinkList  *H)
{  LinkList  *p;
```

```
    printf("\n 所有学生的信息如下:");
    printf("\n 学号    姓名    语文    数学    英语\n");
    p=H->next;
    while(p!=NULL)
    {   printf("%2d%10s%9d%9d%9d\n",p->number,p->name,p->chinese,
        p->math,p->english);
        p=p->next;
    }
}
/*****************主函数*************************/
main()
{   LinkList  *H,*p,*q,x;
    int  i,n,menux,flag,num;
    clrscr();    /*清屏*/
    out();
    H=InitList();
    scanf("%d",&menux);
    do
    {   switch(menux)                  /*用来判断用户要进行何种操作*/
        {  case 0:exit(0); break;      /*退出*/
           case 1:                     /*建立一个学生信息表*/
              printf("\n 请输入要生成的学生信息表的元素个数:");
              scanf("%d",&n);
              CreateList(H,n);
              printf("\n 建立的学生信息表为:\n");
              DisLinkList(H);
              break;
         case 2:                       /*在学生信息表中插入一个元素*/
              printf("\n 请输入要插入的学生位置 ");
              scanf("%d",&i);
              printf("\n 请输入要插入的学生信息:");
              printf("\n 请输入学生的学号：");
              scanf("%d",&x.number);
              printf("\n 请输入学生的姓名：");
              scanf("%s",x.name);
              printf("\n 请输入学生的语文、数学和英语成绩：");
              scanf("%d%d%d",&x.chinese,&x.math,&x.english);
              flag=Ins_List(H,i,x);
              if(flag)
              {    printf("\n 插入后的学生信息表为: ");
                   DisLinkList(H);
              }
              break;
         case 3:                       /*在学生信息表中删除一个元素*/
              printf("\n 请输入要删除的学生的位置:");
              scanf("%d",&i);
              flag=Del_List(H,i,&x);
              if(flag)
              {   printf("\n 删除第%d 个学生后,表中信息为:",i);
                  DisLinkList(H);
              }
              break;
```

```
            case 4:                          /*在学生信息表中查找某学号学生信息*/
                printf("\n 请输入查找的学生学号:\n");
                scanf("%d",&num);
                if((q=Locate(H,num))!=NULL)
                {   printf("\n 在学生信息表中存在着学号为%d 的学生! ",num);
                   printf("\n 学号    姓名    语文   数学   英语\n");
                   printf("%2d%10s%9d%9d%9d",q->number,q->name,q->chinese,
                           q->math,q->english);
                }
                else
                    printf("\n 在学生信息表中不存在着学号为%d 的学生! ",num);
                    break;
                default:
                    printf("\n 输入选项错误,请重新输入(0-4)!");
            }
            out();
            scanf("%d",&menux);
        }while(1);
    }
```

五、程序运行步骤及结果

说明：程序运行自动显示的信息无下画线，用户输入的信息用下画线显示，“↙”表示回车。本程序有主菜单，设有 0～4 五个选项，每次可以输入 0～4 来执行相关操作，并且每次执行完后都会重新显示菜单，当输入 0 时结束该程序。

程序一次运行结果如下：

```
*********************************************
请输入序号（0-4）选择要进行的操作:
0------退出
1------建立一个学生信息表
2------插入一个学生信息
3------删除一个学生信息
4------查找一个是否在该学生信息表中
*********************************************
1↙
请输入要生成的学生信息表的元素个数:2↙
请输入第 1 个学生的学号：1↙
请输入第 1 个学生的姓名：liu↙
请输入第 1 个学生的语文、数学和英语成绩：78  88  89↙
请输入第 2 个学生的学号：2↙
请输入第 2 个学生的姓名：wang↙
请输入第 2 个学生的语文、数学和英语成绩：65  89  93↙
建立的学生信息表为:
学号    姓名     语文    数学     英语
1       liu      78      88       89
2      wang      65      89       93
*********************************************
请输入序号（0-4）选择要进行的操作:
0------退出
1------建立一个学生信息表
2------插入一个学生信息
```

```
3------删除一个学生信息
4------查找一个是否在该学生信息表中
*****************************************
2↙
   请输入要插入的学生位置 3↙
请输入要插入的学生信息:
请输入学生的学号：3↙
请输入学生的姓名：zhao↙
请输入学生的语文、数学和英语成绩：92  75  81↙
插入后的学生信息表为:
学号    姓名     语文    数学     英语
1       liu      78      88       89
2      wang      65      89       93
3      zhao      92      75       81
*****************************************
请输入序号（0-4）选择要进行的操作:
0------退出
1------建立一个学生信息表
2------插入一个学生信息
3------删除一个学生信息
4------查找一个是否在该学生信息表中
*****************************************
   3↙
请输入要删除的学生的位置:2↙
删除第 2 个学生后,表中信息为:
学号    姓名     语文    数学     英语
1       liu      78      88       89
3      zhao      92      75       81
*****************************************
请输入序号（0-4）选择要进行的操作:
0------退出
1------建立一个学生信息表
2------插入一个学生信息
3------删除一个学生信息
4------查找一个是否在该学生信息表中
*****************************************
   4↙
请输入查找的学生学号:3↙
学号    姓名    语文    数学    英语
3      zhao     92      75      81
*****************************************
请输入序号（0-4）选择要进行的操作:
0------退出
1------建立一个学生信息表
2------插入一个学生信息
3------删除一个学生信息
4------查找一个是否在该学生信息表中
*****************************************

0↙
```

附录A　课后习题参考答案

课后习题一参考答案

一、选择题

1．B　2．D　3．C　4．D　5．A　6．C　7．B　8．C　9．C　10．B

二、填空题

1．{ ，} 2．主　3．机器语言，汇编语言，高级语言　4．自顶向下，逐步求精；模块化设计；结构化编程　5．函数，函数　6．.c　，　.obj　，　.exe　7．库函数

课后习题二参考答案

一、选择题

1. A　2. B　3. C　4. D　5. C　6. A　7. C　8. B　9. A　10. C　11. B　12. D　13．B　14．B　15．C

二、填空题

1．1　2．0，2　3．3，E　4．2，2，1

5．4　6．0

7．i=6,x=5↙（“↙”为回车符）

j=6,y=6

三、编程题

1．程序如下：

```
main()
{   int  a;
    printf("\nPlease input a int number:");
    scanf("%d",&a);
    a=a>0?a:-a;
    printf("absulute a is:%d",a);
}
```

2．（1）0　（2）1　（3）13　（4）3　（5）11

3．解：在这个程序中，r、v 是变量，且 r 的值为 2，因此在程序中要对变量 r、v 进行定义，并对 r 进行赋初始值，要注意 r 是整数，v 是小数，□则为常数，在程序中用自定义 PI 表示□，且值为 3.14。源程序代码如下：

```
#include  "stdio.h"
#define  PI  3.14
main()
{   int  r=2;
```

```
    float  v;
    v=3.0/4*PI*r*r*r;
    printf("v=%f",v);
}
```

4. 解：程序代码如下:

```
#include  "stdio.h"
main()
{   int  a=1,b=2,c=3;
    int  t;
    t=(a>b)?a:b;
    t=(t>c)?t:c;
    printf("the max is:%d",t);
}
```

课后习题三参考答案

一、选择题

1. C　2. A　3.B　4. A　5. B　6. C　7. C　8. A　9. A　10. D　11. D　12. B
13. C　14. B　15. C

二、填空题

1. 1　2. a=3,b=2,c=2　3. i=5,j=4,k=6　4. 1024
5. −1　6. 11

三、编程题

1. 源程序如下：

```
main()
{   int  a;
    printf("input data:");
    scanf("%d",&a);
    if(a%2==0)
      printf("%d is odd number!",a);
    else
       printf("%d is even number!",a);
}
```

2. 源程序如下：

```
main()
{   int  n,s=0;
    for(n=1;n<=20;n++)
    if(n%5==0)
    {   s=s+n;
        printf(“%d,”,n);
    }
    printf(“\ns=%d”,s);
}
```

3. 源程序如下：

```
#include  "stdio.h"
void  main()
```

```
{   long  y,n;
    scanf("%ld",&y)
    while(y!=0)
    {   n=y%10;
        printf("%ld",n);
        y=y/10;
    }
}
```

课后习题四参考答案

一、选择题

1. C　2. D　3. D　4. C　5. B　6. C　7. A　8. A　9. D　10. C　11. A　12. B　13. B　14. D　15. C

二、填空题

1. 按行主顺序存放　2. 0，4　3. 11　4. 0
5. CDABC　6. Happy　7. 6　8. [1]j=2;[2]j>=0
9. 10000　10. 1　3　4　5

三、编程题

1. 程序如下：

```
main()
{   int  a[5],max,min,i,x,y;
    printf("\nPlease input 5 integer number:");
    for(i=0;i<5;i++)
      scanf("%d",&a[i]);
    printf("\nThe  array:");
    for(i=0;i<5;i++)
      printf("%5d",a[i]);
    min=a[0];max=a[0];
    for(i=1;i<5;i++)
   {   if(a[i]<min)
         {   min=a[i];x=i;}
       if(a[i]>max)
         {   max=a[i];y=i; }
    }
    a[x]=max;a[y]=min;
    printf("\nThe position of min number is:%3d",y);
    printf("\nThe position of max number is:%3d",x);
    printf("\nThe exchage array:");
    for(i=0;i<5;i++)
      printf("%5d",a[i]);
}
```

2. 程序如下：

```
main()
{   int  a[3][4]={{3,-2,7,5},{1,0,4,-3},{6,8,0,2}};
    int  b[3][4]={{-2,0,1,4},{5,-1,7,6},{6,8,0,2}};
    int  i,j,c[3][4];
```

```
        printf("\nArray a:\n");
        for(i=0;i<3;i++)
       {   for(j=0;j<4;j++)
              printf("%3d",a[i][j]);
           printf("\n");
        }
        printf("\nArray b:\n");
        for(i=0;i<3;i++)
        {   for(j=0;j<4;j++)
              printf("%3d",b[i][j]);
            printf("\n");
        }
         for(i=0;i<3;i++)
           for(j=0;j<4;j++)
              c[i][j]=a[i][j]+b[i][j];
         printf("\nArray c(=Array a + Array b):\n");
         for(i=0;i<3;i++)
        {   for(j=0;j<4;j++)
                printf("%3d",c[i][j]);
            printf("\n");
        }
    }
```

3. 程序如下：

```
#include  "stdio.h"
#include  "string.h"
main()
{   char  string[80],c;
    int  i,num=0,word=0;
    printf("please input a line of words:\n");
    gets(string);
    for(i=0;(c=string[i])!='\0';i++)
       if(c==' ')
          num++;
    printf("there are %d word in the line.\n",num+1);
}
```

课后习题五参考答案

一、选择题

1. A　2. C　3. B　4. B　5. A　6. B　7. D　8. B　9. D　10. B　11. B　12. D　13. D　14. B　15. A

二、填空题

1. 0　10　20　30　40　50　60　70　80　90　2. 8,17

3.【1】age(n−1)+2　【2】age(5)　4.【1】−7 3 5 7 10　【2】冒泡法排序

5. 1　2　3

三、编程题

1. 程序如下：

```
double  xpower(float x, int n)
{  if (n==0)
      return(1);
   else
      return  xpower(x,n)*x;
}
main()
  {  float  x ;
   int  n;
   printf("\n input x : ");
   scanf("%f", &x);
   printf("\n input n : ");
   scanf("%d", &n);
   printf("\nxpower(%f,%d)=%.0lf",x,n,xpower(x,n));
}
```

2. 程序如下：

```
#include  <stdio.h>
fun(int  x,int  y)
{   int  z;
    z=abs(x-y);
    return(z);
}
main()
{   int  a,b,c;
    printf("input a and b:");
    scanf("%d%d",&a,&b);
    c=fun(a,b);
    printf("c=%d",c);
}
```

3. 程序如下：

```
int  sum_day(int  month,int  day),leap(int  year);
main()
{   int  day,month,year,days;
    printf("\ninput year,month,day : \n");
    scanf("%d,%d,%d",&year,&month,&day);
    days=sum_day(month,day);
    if(leap(year)&&month>=3)
    days+=1;
    printf("It is the %dth day.\n",days);
}
int  day_table[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
int  sum_day(int  month,int  day)
{   int  i;
    for(i=1;i<month;i++)
    day+=day_table[i];
    return(day);
}
    int  leap(int  year)
{   int  leap_data;
    leap_data= year%400==0||(year%4==0&&year%100!=0);
    return(leap_data);
}
```

课后习题六参考答案

一、选择题

1. D　2. D　3. B　4. A　5. D　6. D　7. C　8. C　9. B　10. A

二、填空题

1. #define　POW(a,b)　((a)*(b))　　2. 880　　3. 12

4. x=9,y=5　　5. □□□9.0

三、编程题

1. 解：源程序如下：

```
/*求两个整数相除的余数*/
#define  MOD(a,b)  (a%b)
main()
{   int  a,b;
    printf("input  two  integer  a,b:");
    scanf("%d,%d",&a,&b);
    printf("a  mod  b  is:%d\n",MOD(a,b));
}
```

2. 解：源程序如下：

```
#define  swap(x,y) {int  t;t=x;x=y;y=t;}
main()
{   int  i,a[10],b[10];
    for(i=0;i<10;i++)
      scanf("%d",&a[i]);
    for(i=0;i<10;i++)
      scanf("%d",&b[i]);
    for(i=0;i<10;i++)
      swap(a[i],b[i]);
    for(i=0;i<10;i++)
      printf("%5d",a[i]);
    for(i=0;i<10;i++)
      printf ("%5d",b[i]);
}
```

3. 解：源程序如下：

```
define YEAR(y) (((y)%4==0&&(y)%100!=0)||((y)%400==0))
main()
{ int year;
   printf("input year:");
   scanf("%d",&year);
   if(YEAR(year))
      printf("%d is bissextile!\n",year);
   else
      printf("%d isn't bissextile!\n",year);
}
```

4. 解：源程序如下：

```
#include "stdio.h"
#define UPPER 1                                  /*定义符号常量为 1*/
main()
{  char c;
```

```
    int count=0;
    printf("please input text:\n");
    while(1)
    { c=getchar();
      if(c=='\n') break;                              /*输入回车时结束输入*/
      #if  UPPER
        if(c>='A'&&c<='Z')   count++;                 /*求大写字母的个数*/
      #else
        if(c>='a'&&c<='z')    count++;                /*求小写字母的个数*/
      #endif
    }
    #if  UPPER
      printf("count of upper letter:%d\n",count);  /*显示大写字母的个数*/
    #else
      printf("count of less letter:%d\n",count);   /*显示小写字母的个数*/
    #endif
  }
```

课后习题七参考答案

一、选择题

1. C　2. C　3. C　4. B　5. D　6. B　7. C　8. D　9. C　10. C　11. C
12. C　13. C　14. C　15. B　16. A　17. C　18. A　19. C　20. A

二、填空题

1. 2　4　　2. 6　　3. G　F　E　D　C　B　A
4. goodgood　　5. 10　　6. TW&T
7. 16, 19　　8. 931

三、编程题

1. 程序如下：

```
#include  "stdio.h"
#define  M  100
#define  N  100
int   MaxVal(int  arr[][N], int  *m, int  *n);
main()
{   int  array[M][N],i,j,line,col,max;
    printf("\nPlease input column of array: ");
    scanf("%d", &col);
    printf("\nPlease input line of array: ");
    scanf("%d", &line);
    printf("\nPlease input data ,use comma  between data: ");
    for(i=0;i<line;i++)
    for(j=0;j<col;j++)
    scanf("%d", &array[i][j]);
    printf( "\n");
    for(i=0;i<line;i++)
    {   for(j=0;j<col;j++)
        printf( "%6d",array[i][j]);
```

```
            printf( "\n");
        }
        printf( "\n");
        max=MaxVal(array,&line,&col);
        printf( "\nThe maximum value is %d: ",max);
        printf( "\nThe line is %d: ",line);
        printf( "\nThe col  is %d: ",col);
        printf( "\n");
    }
    int  MaxVal(int  arr[][N],int  *m,int  *n)
   {   int  i,j,line=0,col=0;
        int  max=arr[0][0];
        int  (*p)[N];
        p=arr;
        for(i=0;i<*m;i++)
        for(j=0;j<*n;j++)
        if(max<*(*(p+i)+j))
       {   max=*(*(p+i)+j);   line=i;     col=j;  }
       *m=line;
       *n=col;
       return max;
    }
```

2. 程序如下：

```
    #include  "stdio.h"
    main()
    {   /* 声明用于存放运动员号码的数组 */
        int  h[]={1001,1002,1003,1004};
        /* 声明用于存放运动员成绩的数组*/
        float  x[]={12.3f,13.1f,11.9f,12.1f};
        /*声明用于存放运动姓名的字符型指针数组*/
        char  *p[]={"Wang hua","Zhang jian","Li wei","Hua ming"};
        /*i,j,it 用做循环控制变量和临时变量*/
        int  i,j,it;
        /*ft 用做暂存变量*/
        float  ft;
        /*pt 为字符型指针变量,用做暂存指针变量*/
        char  *pt;
        /*用选择法对数组 x 进行排序，并相应调整数组 h 和 p 中的数据*/
        for (i=0;i<=3;i++)
        for (j=i+1;j<=3;j++)
        if (x[i]<=x[j])
        {   ft=x[i];x[i]=x[j];x[j]=ft;
            it=h[i];h[i]=h[j];h[j]=it;
            pt=p[i];p[i]=p[j];p[j]=pt;
        }
        /*以下打印排序结果*/
        for (i=0;i<=3;i++)
        printf("%s%d%s\t%6.2f\t%s  %d\t  %s  %s\n"," 第 ",i+1," 名 :",x[i]," 编
               号:",h[i],"姓名:",p[i]);
    }
```

3. 程序如下：

```
#include "stdio.h"
void search(int (*pa)[4],int n);
main()
{  int n;
   int score[5][4]={{73,86,80,85},{90,68,72,83},{80,75,77,82},
                    {90,89,92,85},{65,74,80,69}};
   printf("\nEnter student NO.(0--4):");
   scanf("%d",&n);
   search(score,n);
}
void search(int (*pa)[4],int n)
{  int i;
   printf("\nThe score of student No.%d are: ",n);
   for(i=0;i<4;i++)
   printf("%6d",*(*(pa+n)+i));
   printf("\n");
}
```

课后习题八参考答案

一、选择题

1. A　2. C　3. D　4. A　5. D　6. D　7. D　8. C　9. D　10. B　11. C　12. C　13. C　14. B　15. C

二、填空题

1. 10,x　2. 2,3　3. sizeof(struct ps) 或 sizeof(bt)

4. 【1】max=person[i].age 【2】min=person[i].age 【3】&&　5. 2,5<回车符>dime,dollar

三、编程题

1. 程序如下：

```
main()
{  struct study
   {  int mid;
      int end;
      int average;
   }math;
   printf("please input 2 integer:");
   scanf("%d%d",&math.mid,&math.end);
   math.average=(math.mid+math.end)/2;
   printf("average=%d\n",math.average);
}
```

2. 程序如下：

```
void partition(unsigned long int num)
{  union a
   {  unsigned int part[2];
      unsigned long int w;
   }n,*p;
   p=&n; n.w=num;
   printf("long int=%lx\n",num);
```

```
    printf("low-partnum=%0x,high-partnum=%0x\n",p->part[0],p->part[1]);
}
main()
{   unsigned  long  int  x;
    x=0x23456789;
    partition(x);
}
```

3. 枚举类型定义如下：

```
enum  money{fenl=1,fen2=2,fen5=5,jiao1=10,jiao2=20,jiao5=50,yuan1=100,
yuan2=200,yuan5=500,yuan10=1000,yuan50=5000,yuan100=10000};
```

课后习题九参考答案

一、选择题

1. C　2. D　3. B　4. C　5. B　6. A　7. A　8. B

二、填空题

1. 00001111　2. 1111　3. 1001
4. 10　5. 100100

三、编程题

1. 程序如下：

```
main()
{   int  a,b,c;
    printf("input a and b:");
    scanf("%x,%x",&a,&b);
    c=(a&0x00ff)<<8|(b&0x00ff);
    printf("c=%x\n",c);
}
```

2. 程序如下：

```
main()
{   int  x,y,z,n;
    printf("input x,n:");
    scanf("%x%d",&x,&n);
    y=x>>(16-n);
    z=x<<n;
    z=z|y;
    printf("output  z:%x\n",z);
}
```

3. 程序如下：

```
main()
{   unsigned  a,b,c,d;
    scanf("%o",&a);
    b=a>>4;
    c=~(~0<<4);
    d=b&c;
    printf("%o,%d\n%o,%d\n",a,a,d,d);
}
```

课后习题十参考答案

一、选择题

1. A　2. B　3. B　4. C　5. C　6. C　7. B　8. C　9. B　10. A　11. D　12. C　13. A　14. D　15. C

二、填空题

1. 顺序，随机　2. 二进制，ASCII　3. 字符，流式

4. n−1，buf 的首地址　5.【1】 "r"　【2】 fgetc(fp)　【3】 count++

三、编程题

1. 源程序如下：

```
#include "stdio.h"
FILE *fp;
main()
{  int p=0,n=0,z=0,temp;
   fp=fopn("number.dat","r");
   {  while(!feof(fp))
      {  fscanf(fp,"%d",&temp);
         if(temp>0)
            p++;
         else if(temp<0)
            n++;
         else
            z++;
      }
      fclose(fp);
      printf("positive:%3d,negtive:%3d,zero:%3d\n",p,n,z);
   }
}
```

2. 源程序如下：

```
#include "stdio.h"
struct student
{  long int num;       /*学号*/
   char name[10];      /*姓名*/
   int age;            /*年龄*/
   char sex;           /*性别*/
   char speiality[20]; /*专业*/
   char addr[40];      /*住址*/
};
FILE *fp;
main()
{  strut student std;
   fp=fopn("student.dat","rb");
   if(fp==NULL)
     printf("file not found!\n");
   else
   {  while(!feof(fp))
      {  fread(&std,sizeof(strut student),1,fp);
```

```
            if(std.num>=970101&&std.num<=970135)
              printf("%ld %s %d %c\n",std.num,std.name,std.age,std.sex);
        }
        fclose(fp);
    }
}
```

课后习题十一参考答案

编程题

1. 解：源程序如下：

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
main()
{   int i,x,y,num,top,bottom;
    int driver=VGA,mode=VGAHI;
    initgraph(&driver,&mode,"\\tc");
    x=360; y=160;
    num=20;
    top=x-30;bottom=y-30;
    for(i=0;i<num;i++)
    { ellipse(250,250,0,360,top,bottom);top-=5; bottom+=5;}
    getchar();
}
```

2. 解：源程序如下：

```
#include<graphics.h>
main()
{   int xcenter,ycenter,radius;
    int graphdriver=EGA,graphmode=1,x,y;
    initgraph(&graphdriver,&graphmode,"\\tc");
    cleardevice();
    xcenter=320;
    ycenter=180;
    radius=110;
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,4);
    settextjustify(CENTER_TEXT,TOP_TEXT);
    outtextxy(320,6,"this is a pie chart");
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
    setfillstyle(SOLID_FILL,RED);
    pieslice(xcenter+10,ycenter-10,0,90,radius);
    settextjustify(LEFT_TEXT,BOTTOM_TEXT);
    outtextxy(410,100,"25%");
    setfillstyle(WIDE_DOT_FILL,GREEN);
    pieslice(xcenter,ycenter,90,135,radius);
    settextjustify(RIGHT_TEXT,BOTTOM_TEXT);
    outtextxy(270,90,"12.5%");
    setfillstyle(INTERLEAVE_FILL,YELLOW);
    settextjustify(RIGHT_TEXT,CENTER_TEXT);
    pieslice(xcenter-10,ycenter,135,225,radius);
```

```
    settextjustify(RIGHT_TEXT,CENTER_TEXT);
    outtextxy(190,180,"%25");
    setfillstyle(HATCH_FILL,BLUE);
    pieslice(xcenter,ycenter,225,360,radius);
    settextjustify(LEFT_TEXT,TOP_TEXT);
    outtextxy(370,270,"37.5%");
    getch();
    closegraph();
}
```

3. 解：源程序如下：

```
#include <graphics.h>
#define  LEFT 0
#define  TOP   0
#define  RIGHT  639
#define  BOTTOM  479
#define  LINES  400
#define  MAXCOLOR 15
main()
{   int driver,mode,error;
    int x1,y1;
    int x2,y2;
    int dx1,dy1,dx2,dy2;
    int count=0,color=0;
    driver=VGA;
    mode=VGAHI;
    initgraph(&driver,&mode,"\\tc");
    x1=x2=y1=y2=10;
    dx1=dy1=2;  dx2=dy2=3;
    while(!kbhit())
    {    line(x1,y1,x2,y2);
        x1+=dx1;y1+=dy1;x2+=dx2;y2+=dy2;
        if(x1<=LEFT||x1>=RIGHT)    dx1=-dx1;
        if(y1<=TOP||y1>=BOTTOM)    dy1=-dy1;
        if(x2<=LEFT||x2>=RIGHT)    dx2=-dx2;
        if(y2<=TOP||y2>=BOTTOM)    dy2=-dy2;
        if(++count>LINES)
       {   setcolor(color);
           color=(color>=MAXCOLOR)?0:++color;
        }
    }
    closegraph();
}
```

附录B　常用字符与ASCII码对照表

ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	\|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

上表中的控制字符含义如下：

NUL	空	VT	垂直制表	SYN	空转同步
SOH	标题开始	FF	走纸控制	ETB	信息组传送结束
STX	正文开始	CR	回车	CAN	作废
ETX	正文结束	SO	移位输出	EM	纸尽
EOY	传输结束	SI	移位输入	SUB	换置
ENQ	询问字符	DLE	空格	ESC	换码
ACK	承认	DC1	设备控制 1	FS	文字分隔符
BEL	报警	DC2	设备控制 2	GS	组分隔符
BS	退一格	DC3	设备控制 3	RS	记录分隔符
HT	横向列表	DC4	设备控制 4	US	单元分隔符
LF	换行	NAK	否定	DEL	删除

附录C　运算符的优先级、结合方向及口诀

优　先　级	运　算　符	名　　称	运算符类型	结 合 方 式
1	() [] -> .	括号（函数等） 数组下标运算符 结构体成员运算符 结构体成员运算符		由左向右
2	! ~ ++ −− + − (类型) * & sizeof	逻辑非运算符 按位取反运算符 自增运算符 自减运算符 正号运算符 负号运算符 强制类型转换 指针运算符 取地址运算符 长度运算符	单目运算符	由右向左
3	* / %	乘法运算符 除法运算符 取模运算符	算术（双目）运算符	由左向右
4	+ −	加法运算符 减法运算符	算术（双目）运算符	由左向右
5	<< >>	左移运算符 右移运算符	位（双目）运算符	由左向右
6	< <= >= >	小于运算符 小于等于运算符 大于等于运算符 大于运算符	关系（双目）运算符	由左向右
7	== !=	等于运算符 不等于运算符	关系（双目）运算符	由左向右
8	&	按位与运算符	位（双目）运算符	由左向右
9	^	按位异或运算符	位（双目）运算符	由左向右
10	\|	按位或运算符	位（双目）运算符	由左向右
11	&&	逻辑与运算符	逻辑（双目）运算符	由左向右
12	\|\|	逻辑或运算符	逻辑（双目）运算符	由左向右
13	? :	条件运算符	条件（三目）运算符	由右向左
14	= += −= *= /= &= ^= \|= <<= >>=	各种赋值运算符	赋值（双目）运算符	由右向左
15	,	逗号运算符		由左向右

C 语言运算符的优先级汇总表口诀：

口　　诀	含　　义
括号成员第一	括号运算符[]() 成员运算符.和->
全体单目第二	所有的单目运算符比如++ -- +（正） -（负） 指针运算*&
乘除余三，加减四	这个“余”是指取余运算即%
移位五，关系六	移位运算符：<< >> ，关系：> < >= <= 等
等于（与）不等排第七	即== !=
位与异或和位或	这几个都是位运算: 位与（&）异或（^）位或（\|）
“三分天下”八九十	
逻辑或跟与	逻辑运算符:\|\| 和 &&
十二和十一	注意顺序:优先级（\|\|） 底于 优先级（&&）
条件高于赋值	三目运算符优先级排到 13 位只比赋值运算符和“,”高， 需要注意的是赋值运算符很多！
逗号运算级最低	逗号运算符优先级最低

参 考 文 献

［1］谭浩强．C 语言程序设计．北京：清华大学出版社，1991
［2］王成瑞．C 语言程序设计．北京：中国水利水电出版社，2006
［3］闫利平，马林艺，高贞．C 语言基础教程．北京：电子工业出版社，2000
［4］李春葆．C 语言与习题解答．北京：清华大学出版社，1999
［5］田淑清．全国计算机等级考试（二级教程）——C 语言程序设计．北京：高等教育出版社，2006

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888
传　　真：（010）88254397
E-mail：　dbqq@phei.com.cn
通信地址：北京市万寿路 173 信箱
　　电子工业出版社总编办公室
邮　　编：100036